

**MC680x0**

**COLLABORATORS**

	<i>TITLE :</i> MC680x0		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		July 28, 2024	

**REVISION HISTORY**

<i>NUMBER</i>	<i>DATE</i>	<i>DESCRIPTION</i>	<i>NAME</i>

# Contents

<b>1</b>	<b>MC680x0</b>	<b>1</b>
1.1	MC680x0 Reference 1.0, ©April 1995 by Flint/DARKNESS. . . . .	1
1.2	MC680x0 Reference Copyright . . . . .	1
1.3	MC680x0 Reference: contacting the author . . . . .	2
1.4	MC68000 Instructions timing . . . . .	2
1.5	MC68030 Instructions timing . . . . .	3
1.6	68030 Instruction times . . . . .	3
1.7	Fetch Effective Address (FEA) . . . . .	8
1.8	Fetch Immediate Effective Address (FIEA) . . . . .	9
1.9	Calculate Effective Address (CEA) . . . . .	11
1.10	Calculate Immediate Effective Address (CIEA) . . . . .	12
1.11	Jump Effective Address (JEA) . . . . .	13
1.12	Some notes about 020+ . . . . .	14
1.13	Arithmetics Instructions . . . . .	15
1.14	Logic Instructions . . . . .	15
1.15	Shift and Rotate Instructions . . . . .	16
1.16	Bit Manipulation Instructions . . . . .	16
1.17	Bit Field Manipulation Instructions . . . . .	16
1.18	BCD Instructions . . . . .	16
1.19	Datas tranfert Instructions . . . . .	16
1.20	Flow Control Instructions . . . . .	17
1.21	Privileged Instructions . . . . .	17
1.22	'Exceptions' Instructions . . . . .	17
1.23	CCR related Instructions . . . . .	18
1.24	PMMU control Instructions . . . . .	18
1.25	Multiprocessor Instructions . . . . .	18
1.26	CoProcessor Instructions . . . . .	18
1.27	Alphabetical Index . . . . .	18
1.28	Add Binary Coded Decimal (w/extend) . . . . .	21
1.29	ADD integer . . . . .	22

---

1.30	ADD Address . . . . .	23
1.31	ADD Immediate . . . . .	24
1.32	ADD 3-bit immediate Quick . . . . .	26
1.33	ADD integer with eXtend . . . . .	27
1.34	Logical AND . . . . .	28
1.35	Logical AND Immediate . . . . .	30
1.36	Logical AND Immediate to CCR . . . . .	31
1.37	Logical AND Immediate to SR (privileged) . . . . .	32
1.38	Arithmetic Shift Left and Arithmetic Shift Right . . . . .	32
1.39	Conditional branch . . . . .	35
1.40	Bit CHanGe . . . . .	35
1.41	Bit CLeaR . . . . .	37
1.42	Bit Field CHanGe . . . . .	38
1.43	Bit Field CLeaR . . . . .	40
1.44	Bit Field Signed EXtract . . . . .	41
1.45	Bit Field Unsigned EXtract . . . . .	43
1.46	Bit Field Find First One set . . . . .	44
1.47	Bit Field INSert . . . . .	46
1.48	Bit Field SET . . . . .	47
1.49	Bit Field TeST . . . . .	48
1.50	BreaK-PoinT . . . . .	50
1.51	Unconditional BRAnch . . . . .	51
1.52	Bit SET . . . . .	52
1.53	Branch to SubRoutine . . . . .	53
1.54	Bit TeST . . . . .	54
1.55	CALL Module . . . . .	56
1.56	Compare And Swap . . . . .	56
1.57	Compare And Swap (two-operand) . . . . .	57
1.58	CHeCK bounds . . . . .	59
1.59	CHeCK register against upper and lower bounds . . . . .	60
1.60	CLeaR . . . . .	61
1.61	CoMPare . . . . .	62
1.62	CoMPare register against upper and lower bounds . . . . .	64
1.63	CoMPare Address . . . . .	65
1.64	CoMPare Immediate . . . . .	66
1.65	CoMPare Memory . . . . .	68
1.66	Branch on CoProcessor condition . . . . .	68
1.67	Decrement and Branch on CoProcessor condition . . . . .	69
1.68	GENeral CoProcessor intruction . . . . .	70

---

1.69 RESTORE CoProcessor instruction (PRIVILEGED)	71
1.70 SAVE CoProcessor instruction (PRIVILEGED)	72
1.71 Set one byte on CoProcessor condition	73
1.72 Exception generation on CoProcessor condition	74
1.73 Decrement and Branch Conditionally	75
1.74 Signed DIVide	77
1.75 Unsigned DIVide	79
1.76 Exclusive logical OR	81
1.77 Exclusive OR Immediate	83
1.78 Exclusive OR Immediate to CCR	84
1.79 Exclusive OR immediated with SR (PRIVILEGED)	84
1.80 Register EXchanGe	85
1.81 Sign EXTend	86
1.82 Illegal processor instruction	87
1.83 Unconditional far JuMP	87
1.84 Jump to far SubRoutine	88
1.85 Load Effective Address	89
1.86 Create local stack frame	90
1.87 Logical Shift Left and Logical Shift Right	92
1.88 Move Source -> Destination	94
1.89 Move Address Source -> Destination	95
1.90 CCR -> Destination	97
1.91 Source -> CCR	98
1.92 Move from SR (privileged)	99
1.93 Move to SR (PRIVILEGED)	100
1.94 Move to/from USP (privileged)	101
1.95 Move to/from control register	101
1.96 MOVE Multiple registers	102
1.97 MOVE Peripheral data	105
1.98 MOVE signed 8-bit data Quick	106
1.99 MOVE address Space (PRIVILEGED)	106
1.100 Signed and Unsigned MULTIply	108
1.101 Negate Binary Coded Decimal with extend	110
1.102 neg	111
1.103 NEGate with eXtend	112
1.104 No OPeration	113
1.105 Logical complement	114
1.106 Logical OR	115
1.107 Logical OR Immediate	116

1.108	Logical OR immediate to CCR . . . . .	118
1.109	Logical OR immediated to SR (PRIVILEGED) . . . . .	118
1.110	PACK binary coded decimal . . . . .	119
1.111	Push Effective Address . . . . .	120
1.112	Invalidate one or several entries in the ATC (PRIVILEGED) . . . . .	121
1.113	LOAD of an entry in the ATC (PRIVILEGED) . . . . .	122
1.114	MOVE from or to PMMU registers (PRIVILEGED) . . . . .	124
1.115	TESTs a logic address (PRIVILEGED) . . . . .	126
1.116	RESET external devices . . . . .	128
1.117	RESET external devices . . . . .	129
1.118	ROtate Left and ROtate Right . . . . .	129
1.119	ROtate Left with eXtend and ROtate Right with eXtend . . . . .	131
1.120	ReTurn and Deallocate parameter stack frame . . . . .	133
1.121	ReTurn from Exception (PRIVILEGED) . . . . .	134
1.122	ReTurn from process Module . . . . .	134
1.123	ReTurn and Restore CCR . . . . .	135
1.124	ReTurn from Subroutine . . . . .	135
1.125	Subtract Binary Coded Decimal with extend . . . . .	135
1.126	Conditional Set . . . . .	137
1.127	Stop processor execution (PRIVILEGED) . . . . .	138
1.128	SUBtract . . . . .	138
1.129	SUBtract Address . . . . .	140
1.130	SUBtract Immediate . . . . .	141
1.131	SUBtract 3-bit immediate Quick . . . . .	142
1.132	SUBtract with eXtend . . . . .	144
1.133	SWAP register upper and lower words . . . . .	145
1.134	Test And Set operand . . . . .	145
1.135	Initiate processor TRAP . . . . .	147
1.136	Conditional trap . . . . .	147
1.137	Trap on oVerflow . . . . .	148
1.138	TeST operand for zero . . . . .	148
1.139	Free stack frame created by LINK . . . . .	150
1.140	Unpack binary coded decimal . . . . .	150
1.141	Move Instruction Execution Times . . . . .	151
1.142	Standard Instruction Execution Times . . . . .	152
1.143	Immediate Instruction Execution Times . . . . .	153
1.144	Single Operand Instruction Execution Times . . . . .	153
1.145	Rotate Instruction Execution Times . . . . .	154
1.146	Bit Manipulation Instruction Execution Times . . . . .	154

---

1.147	Specification Instruction Execution Times . . . . .	155
1.148	JMP, JSR, LEA, PEA and MOVEM Instruction Execution Times . . . . .	155
1.149	Multi-Precision Instruction Execution Times . . . . .	156
1.150	Miscellaneous Instruction Execution Times . . . . .	157
1.151	Move Peripheral Instruction Execution Times . . . . .	157
1.152	Exception Processing Execution Times . . . . .	158
1.153	ASP68K PROJECT, Sixth Edition . . . . .	158
1.154	Generic Assembler Documentation . . . . .	180

---

# Chapter 1

## MC680x0

### 1.1 MC680x0 Reference 1.0, ©April 1995 by Flint/DARKNESS.

MC 680x0 Reference 1.0  
 ~~~~~  
 ©April/May 1995 by Flint/DARKNESS.

|                          |                                |
|--------------------------|--------------------------------|
| Generic Asm Doc<br>~~~~~ | MC680x0 Optimizations<br>~~~~~ |
|--------------------------|--------------------------------|

MC680x0 Instruction types:  
 ~~~~~

- Arithmetics Instructions
- Logic Instructions
- Shift and Rotate Instructions
- Bit Manipulation Instructions
- Bit Field Manipulation Instructions
- BCD Instructions
  - Datas transfert Instructions
  - Flow Control Instructions
- Privileged Instructions
- 'Exceptions' Instructions
  - CCR related Instructions
- PMMU control Instructions
- Multiprocessor Instructions
- CoProcessor Instructions

Alphabetical Index

MC68000 Instructions timing  
 ~~~~~

MC68030 Instructions timing  
 ~~~~~

Copyright and Distribution  
 ~~~~~

Contacting the author  
 ~~~~~

### 1.2 MC680x0 Reference Copyright

## Copyright and Distribution

This documentation is Copyright ©1995 Flint/DKS. All Rights reserved.  
MC680x0 Reference is freely redistributable. You're permitted to modify it for personal use, but any modifications made must NOT be distributed. If you have made changes you think others would like, send them to me and I'll include them in future versions.

This doc comes with NO WARRANTIES. The author is NOT responsible for any mistakes, or wrong information; the user takes all such responsibility.

No charges may be made for MC680x0 Reference, other than a nominal copy fee. It may NOT be distributed with a commercial product without the author prior consent.

Although MC680x0 Reference is freeware, DONATIONS WOULD BE GLADLY ACCEPTED, either money or stuff you've written yourself. See Contacting the Author.

## 1.3 MC680x0 Reference: contacting the author

### Contacting the author

I can be reached for comments, suggestions, bug reports, money etc. at the following address:

Flint/DKS  
Bruno COSTE  
80, av. de la Lanterne  
Le Castiglione B  
06200 NICE  
FRANCE

Or By E-Mail (it's the account of a friend so specify for Flint/DKS):  
pintaric@samo.unice.fr

## 1.4 MC68000 Instructions timing

To calculate the timings of most 68000 instructions, you will need to first find the number of cycles used by the addressing mode in the table below ('Effective Address Operand Calculation Timing') and then the timing for the actual instruction in the appropriate table.

Move Instruction Execution Times  
Standard Instruction Execution Times  
Immediate Instruction Execution Times  
Single Operand Instruction Execution Times  
Rotate Instruction Execution Times  
Bit Manipulation Instruction Execution Times  
Specification Instruction Execution Times  
JMP, JSR, LEA, PEA and MOVEM Instruction Execution Times  
Multi-Precision Instruction Execution Times  
Miscellaneous Instruction Execution Times  
Move Peripheral Instruction Execution Times

## Exception Processing Execution Times

## Effective Address Operand Calculation Timing

This table lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words, the address computation, and fetching of the memory operand. The number of bus read and write cycles is shown in parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

## Effective Address Calculation Times

	register	Byte,Word Long		
Dn	data register direct		0 (0/0)	0 (0/0)
An	address register direct		0 (0/0)	0 (0/0)
memory				
(An)	address register indirect		4 (1/0)	8 (2/0)
(An)+	address register indirect with post-increment		4 (1/0)	8 (2/0)
-(An)	address register indirect with predec.		6 (1/0)	10 (2/0)
d(An)	address register indirect with displacement		8 (2/0)	12 (3/0)
d(An,ix)	address register indirect with index		10 (2/0)	14 (3/0)
xxx.W	absolute short		8 (2/0)	12 (3/0)
xxx.L	absolute long		12 (3/0)	16 (4/0)
d(PC)	program counter with displacement		8 (2/0)	12 (3/0)
d(PC,ix)	program counter with index		10 (2/0)	14 (3/0)
#xxx	immediate		4 (1/0)	8 (2/0)

The size of the index register (ix) does not affect execution time

## 1.5 MC68030 Instructions timing

68030 Instruction times

Calculate Effective Address (CEA)

Calculate Immediate Effective Address (CIEA)

Fetch Effective Address (FEA)

Fetch Immediate Effective Address (FIEA)

Jump Effective Address (JEA)

Some notes about 020+

## 1.6 68030 Instruction times

- # - Add Calculate Effective Address time (CEA)
- % - Add Calculate Immediate Effective Address time (CIEA)
- \* - Add Fetch Effective Address time (FEA)
- ! - Add Fetch Immediate Effective Address time (FIEA)
- j - Add Jump Effective Address Time (JEA)

(r/p/w) r - Read Cycles  
 p - Prefetch Cycles  
 w - Write Cycles

	Head	Tail	I-Cache	No-Cache
move Rn,Dn	2 0	2(0/0/0)	2(0/1/0)	
move Rn,An	2 0	2(0/0/0)	2(0/1/0)	
*move EA,An	0 0	2(0/0/0)	2(0/1/0)	
*move EA,Dn	0 0	2(0/0/0)	2(0/1/0)	
move Rn,(An)	0 1	3(0/0/1)	4(0/1/1)	
*move SOURCE,(An)	2 0	4(0/0/1)	5(0/1/1)	
move Rn,(An)+	0 1	3(0/0/1)	4(0/1/1)	
*move SOURCE,(An)+	2 0	4(0/0/1)	5(0/1/1)	
move Rn,-(An)	0 2	4(0/0/1)	4(0/1/1)	
*move SOURCE,-(An)	2 0	4(0/0/1)	5(0/1/1)	
*move EA,(d16,An)	2 0	4(0/0/1)	5(0/1/1)	
*move EA,xxx.W	2 0	4(0/0/1)	5(0/1/1)	
*move EA,xxx.L	0 0	6(0/0/1)	7(0/2/1)	
*move EA,(d8,An,Xn)	4 0	6(0/0/1)	7(0/1/1)	
*move EA,(d16,An)	2 0	8(0/0/1)	9(0/2/1)	
*move EA,(d16,PC)	2 0	8(0/0/1)	9(0/2/1)	
*move EA,(d16,An,Xn)	2 0	8(0/0/1)	9(0/2/1)	
*move EA,(d16,PC,Xn)	2 0	8(0/0/1)	9(0/2/1)	
*move EA,([d16,An],Xn)	2 0	10(1/0/1)	11(1/2/1)	
*move EA,([d16,PC],Xn)	2 0	10(1/0/1)	11(1/2/1)	
*move EA,([d16,An],d16)	2 0	12(1/0/1)	14(1/2/1)	
*move EA,([d16,PC],d16)	2 0	12(1/0/1)	14(1/2/1)	
*move EA,([d16,An],d32)	2 0	14(1/0/1)	16(1/3/1)	
*move EA,([d16,PC],d32)	2 0	14(1/0/1)	16(1/3/1)	
*move EA,([d16,An],Xn,d32)	2 0	14(1/0/1)	16(1/3/1)	
*move EA,([d16,PC],Xn,d32)	2 0	14(1/0/1)	16(1/3/1)	
*move EA,(B)	4 0	8(0/0/1)	9(0/1/1)	
*move EA,(d16,B)	4 0	10(0/0/1)	12(0/2/1)	
*move EA,(d32,B)	4 0	14(0/0/1)	16(0/2/1)	
*move EA,([B])	4 0	10(1/0/1)	11(1/1/1)	
*move EA,([B].l)	4 0	10(1/0/1)	11(1/1/1)	
*move EA,([B],d16)	4 0	12(1/0/1)	14(1/2/1)	
*move EA,([B].l,d16)	4 0	12(1/0/1)	14(1/2/1)	
*move EA,([B],d32)	4 0	14(1/0/1)	16(1/2/1)	
*move EA,([B].l,d32)	4 0	14(1/0/1)	16(1/2/1)	
*move EA,([d16,B])	4 0	12(1/0/1)	14(1/2/1)	
*move EA,([d16,B].l)	4 0	12(1/0/1)	14(1/2/1)	
*move EA,([d16,B],d16)	4 0	14(1/0/1)	17(1/2/1)	
*move EA,([d16,B].l,d16)	4 0	14(1/0/1)	17(1/2/1)	
*move EA,([d16,B],d32)	4 0	16(1/0/1)	19(1/3/1)	
*move EA,([d16,B].l,d32)	4 0	16(1/0/1)	19(1/3/1)	
*move EA,([d32,B])	4 0	16(1/0/1)	18(1/2/1)	
*move EA,([d32,B].l)	4 0	16(1/0/1)	18(1/2/1)	
*move EA,([d32,B],d16)	4 0	18(1/0/1)	21(1/3/1)	
*move EA,([d32,B].l,d16)	4 0	18(1/0/1)	21(1/3/1)	
*move EA,([d32,B],d32)	4 0	18(1/0/1)	23(1/3/1)	
*move EA,([d32,B].l,d32)	4 0	18(1/0/1)	23(1/3/1)	
exg Ry,Rx	4 0	4(0/0/0)	4(0/1/0)	
movec Cr,Rn	6 0	6(0/0/0)	6(0/1/0)	
movec Rn,usp vbr caar msp isp	6 0	6(0/0/0)	6(0/1/0)	
movec Rn,sfc dfc cacr	4 0	12(0/0/0)	12(0/1/0)	

```

move ccr,Dn      2 0 4(0/0/0) 4(0/1/0)
#move ccr,mem    2 0 4(0/0/1) 5(0/1/1)
move Dn,ccr      4 0 4(0/0/1) 4(0/1/1)
#move EA,ccr     0 0 4(0/0/1) 4(0/1/1)
move sr,Dn       2 0 4(0/0/1) 4(0/1/1)
#move sr,mem     2 0 4(0/0/1) 4(0/1/1)
*move EA,sr      0 0 8(0/0/0) 10(0/1/0)
%movem EA,r1(+)  2 0 8+4n(n/0/0) 8+4n(n/1/0)
%movem r1,EA(+)  2 0 4+2n(0/0/n) 4+2n(0/1/n)
movep.w Dn,(d16,An) 4 0 10(0/0/2) 10(0/1/2)
movep.w (d16,An),Dn 2 0 10(2/0/0) 10(2/1/0)
movep.l Dn,(d16,An) 4 0 14(0/0/4) 14(0/1/4)
movep.l (d16,An),Dn 2 0 14(4/0/0) 14(4/1/0)
%moves EA,Rn     3 0 7(1/0/0) 7(1/1/0)
%moves Rn,EA     2 1 5(0/0/1) 6(0/1/1)
move usp,An      4 0 4(0/0/0) 4(0/1/1)
move An,usp      4 0 4(0/0/0) 4(0/1/0)
swap Dn          4 0 4(0/0/0) 4(0/1/0)
add Rn,Dn        2 0 2(0/0/0) 2(0/1/0)
adda.w Rn,An     4 0 4(0/0/0) 4(0/1/0)
adda.l Rn,An     2 0 2(0/0/0) 2(0/1/0)
*add EA,Dn       0 0 2(0/0/0) 2(0/1/0)
*adda.w EA,An    0 0 4(0/0/0) 4(0/1/0)
*adda.l EA,An    0 0 2(0/0/0) 2(0/1/0)
*add Dn,EA       0 1 3(0/0/1) 4(0/1/1)
and Dn,Dn        2 0 2(0/0/0) 2(0/1/0)
*and EA,Dn       0 0 2(0/0/0) 2(0/1/0)
*and Dn,EA       0 1 3(0/0/1) 4(0/1/1)
eor Dn,Dn        2 0 2(0/0/0) 2(0/1/0)
*eor Dn,EA       0 1 3(0/0/1) 4(0/1/1)
or Dn,Dn         2 0 2(0/0/0) 2(0/1/0)
or EA,Dn         0 0 2(0/0/0) 2(0/1/0)
or Dn,EA         0 1 3(0/0/1) 4(0/1/1)
sub Rn,Dn        2 0 2(0/0/0) 2(0/1/0)
*sub EA,Dn       0 0 2(0/0/0) 2(0/1/0)
*sub Dn,EA       0 1 3(0/0/1) 4(0/1/1)
suba.w Rn,An     4 0 4(0/0/0) 4(0/1/0)
suba.l Rn,An     2 0 2(0/0/0) 2(0/1/0)
*suba.w EA,An    0 0 4(0/0/0) 4(0/1/0)
*suba.l EA,An    0 0 2(0/0/0) 2(0/1/0)
cmp Rn,Dn        2 0 2(0/0/0) 2(0/1/0)
*cmp EA,Dn       0 0 2(0/0/0) 2(0/1/0)
cmpa Rn,An       4 0 4(0/0/0) 4(0/1/0)
*cmpa EA,An      0 0 4(0/0/0) 4(0/1/0)
!cmp2 EA,Rn(max) 2 0 20(1/0/0) 20(1/1/0)
*muls.w EA,Dn(max) 2 0 28(0/0/0) 28(0/1/0)
!muls.l EA,Dn(max) 2 0 44(0/0/0) 44(0/1/0)
*mulu.w EA,Dn(max) 2 0 28(0/0/0) 28(0/1/0)
!mulu.l EA,Dn(max) 2 0 44(0/0/0) 44(0/1/0)
divs.w Dn,Dn(max) 2 0 56(0/0/0) 56(0/1/0)
*divs.w EA,Dn(max) 0 0 56(0/0/0) 56(0/1/0)
!divs.l Dn,Dn(max) 6 0 90(0/0/0) 90(0/1/0)
!divs.l EA,Dn(max) 0 0 90(0/0/0) 90(0/1/0)
divu.w Dn,Dn(max) 2 0 44(0/0/0) 44(0/1/0)
*divu.w EA,Dn(max) 0 0 44(0/0/0) 44(0/1/0)
!divu.l Dn,Dn(max) 6 0 78(0/0/0) 78(0/1/0)
!divu.l EA,Dn(max) 0 0 78(0/0/0) 78(0/1/0)

```

```

moveq #(data),Dn      2 0 2(0/0/0) 2(0/1/0)
addq #(data),Rn       2 0 2(0/0/0) 2(0/1/0)
*addq #(data),Mem     0 1 3(0/0/1) 4(0/1/1)
subq #(data),Rn       2 0 2(0/0/0) 2(0/1/0)
*subq #(data),Mem     0 1 3(0/0/1) 4(0/1/1)
!addi #(data),Dn      2 0 2(0/0/0) 2(0/1/0)
!addi #(data),Mem     0 1 3(0/0/1) 4(0/1/1)
!andi #(data),Dn      2 0 2(0/0/0) 2(0/1/0)
!andi #(data),Mem     0 1 3(0/0/1) 4(0/1/1)
!eori #(data),Dn      2 0 2(0/0/0) 2(0/1/0)
!eori #(data),Mem     0 1 3(0/0/1) 4(0/1/1)
!ori #(data),Dn       2 0 2(0/0/0) 2(0/1/0)
!ori #(data),Mem     0 1 3(0/0/1) 4(0/1/1)
!subi #(data),Dn      2 0 2(0/0/0) 2(0/1/0)
!subi #(data),Mem     0 1 3(0/0/1) 4(0/1/1)
!cmpi #(data),Dn      2 0 2(0/0/0) 2(0/1/0)
!cmpi #(data),Mem     0 0 3(0/0/1) 2(0/1/0)
abcd Dn,Dn           0 0 4(0/0/0) 4(0/1/0)
abcd -(An),-(An)     2 1 13(2/0/1) 14(2/1/1)
sbcd Dn,Dn           0 0 4(0/0/0) 4(0/1/0)
sbcd -(An),-(An)     2 1 13(2/0/1) 14(2/1/1)
addx Dn,Dn           2 0 2(0/0/0) 2(0/1/0)
addx -(An),-(An)     2 1 9(2/0/1) 10(2/1/1)
subx Dn,Dn           2 0 2(0/0/0) 2(0/1/0)
subx -(An),-(An)     2 1 9(2/0/1) 10(2/1/1)
cmpm (An)+,(An)+     0 0 8(2/0/0) 8(2/1/0)
pack Dn,Dn,#(data)   6 0 6(0/0/0) 6(0/1/0)
pack -(An),-(An),#(data) 2 1 11(1/0/1) 11(1/1/1)
unpk Dn,Dn,#(data)   8 0 8(0/0/0) 8(0/1/0)
unpk -(An),-(An),#(data) 2 1 11(1/0/1) 11(1/1/1)
clr Dn               2 0 2(0/0/0) 2(0/1/0)
#clr Mem             0 1 3(0/0/1) 4(0/1/1)
neg Dn               2 0 2(0/0/0) 2(0/1/0)
*neg Mem            0 1 3(0/0/1) 4(0/1/1)
negx Dn              2 0 2(0/0/0) 2(0/1/0)
*negx Mem           0 1 3(0/0/1) 4(0/1/1)
not Dn               2 0 2(0/0/0) 2(0/1/0)
*not Mem            0 1 3(0/0/1) 4(0/1/1)
ext Dn               4 0 4(0/0/0) 4(0/1/0)
nbcd Dn              0 0 6(0/0/0) 6(0/1/0)
scc Dn               4 0 4(0/0/0) 4(0/1/0)
#scc Mem            0 1 5(0/0/1) 5(0/1/1)
tas Dn               4 0 4(0/0/0) 4(0/1/0)
#tas Mem            3 0 12(1/0/1) 12(1/1/1)
tst Dn               0 0 2(0/0/0) 2(0/1/0)
*tst Mem            0 0 2(0/0/0) 2(0/1/0)
ls? #(data),Dy       4 0 4(0/0/0) 4(0/1/0)
ls? Dx,Dy (shift << size) 6 0 6(0/0/0) 6(0/1/0)
ls? Dx,Dy (shift >> size) 8 0 8(0/0/0) 8(0/1/0)
*ls? Mem (by one)    0 0 4(0/0/1) 4(0/1/1)
asl #(data),Dy       2 0 6(0/0/0) 6(0/1/0)
asl Dx,Dy            4 0 8(0/0/0) 8(0/1/0)
asl Mem (by one)     0 0 6(0/0/0) 6(0/1/0)
asr #(data),Dy       4 0 4(0/0/0) 4(0/1/0)
asr Dx,Dy (shift << size) 6 0 6(0/0/0) 6(0/1/0)
asr Dx,Dy (shift >> size) 10 0 10(0/0/0) 10(0/1/0)
*asr Mem (by one)    0 0 4(0/0/0) 4(0/1/0)

```

```

ro?  #(data),Dy    4 0 6(0/0/0) 6(0/1/0)
ro?  Dx,Dy        6 0 8(0/0/0) 8(0/1/0)
*ro? Mem (by one)  0 0 6(0/0/1) 6(0/1/1)
rox? Dn          10 0 12(0/0/0) 12(0/1/0)
*rox? Mem (by one)  0 0 4(0/0/0) 4(0/1/0)
btst #(data),Dn   4 0 4(0/0/0) 4(0/1/0)
btst Dn,Dn        4 0 4(0/0/0) 4(0/1/0)
!btst #(data),Mem 0 0 4(0/0/0) 4(0/1/0)
*btst Dn,Mem      0 0 4(0/0/0) 4(0/1/0)
bchg #(data),Dn   6 0 6(0/0/0) 6(0/1/0)
bchg Dn,Dn        6 0 6(0/0/0) 6(0/1/0)
!bchg #(data),Mem 0 0 6(0/0/1) 6(0/1/1)
*bchg Dn,Mem      0 0 6(0/0/1) 6(0/1/1)
bclr #(data),Dn   6 0 6(0/0/0) 6(0/1/0)
bclr Dn,Dn        6 0 6(0/0/0) 6(0/1/0)
!bclr #(data),Mem 0 0 6(0/0/1) 6(0/1/1)
*bclr Dn,Mem      0 0 6(0/0/1) 6(0/1/1)
bset #(data),Dn   6 0 6(0/0/0) 6(0/1/0)
bset Dn,Dn        6 0 6(0/0/0) 6(0/1/0)
!bset #(data),Mem 0 0 6(0/0/1) 6(0/1/1)
%bset Dn,Mem      0 0 6(0/0/1) 6(0/1/1)
bftst Dn          8 0 8(0/0/0) 8(0/1/0)
%bftst Mem (< 5 bytes) 6 0 10(1/0/0) 10(1/1/0)
%bftst Mem (> 5 bytes) 6 0 14(2/0/0) 14(2/1/0)
bfchg Dn          14 0 14(0/0/0) 14(0/1/0)
%bfchg Mem (< 5 bytes) 6 0 14(1/0/1) 14(1/1/1)
%bfchg Mem (> 5 bytes) 6 0 22(2/0/2) 22(2/1/2)
bfclr Dn          14 0 14(0/0/0) 14(0/1/0)
%bfclr Mem (< 5 bytes) 6 0 14(1/0/1) 14(1/1/1)
%bfclr Mem (> 5 bytes) 6 0 22(2/0/2) 22(2/1/2)
bfset Dn          14 0 14(0/0/0) 14(0/1/0)
%bfset Mem (< 5 bytes) 6 0 14(1/0/1) 14(1/1/1)
%bfset Mem (> 5 bytes) 6 0 22(2/0/2) 22(2/1/2)
bfexts Dn         10 0 10(0/0/0) 10(0/1/0)
%bfexts Mem (< 5 bytes) 6 0 12(1/0/0) 12(1/1/0)
%bfexts Mem (> 5 bytes) 6 0 18(2/0/0) 18(2/1/0)
bfextu Dn         10 0 10(0/0/0) 10(0/1/0)
%bfextu Mem (< 5 bytes) 6 0 12(1/0/0) 12(1/1/0)
%bfextu Mem (> 5 bytes) 6 0 18(2/0/0) 18(2/1/0)
bfins Dn          12 0 12(0/0/0) 12(0/1/0)
%bfins Mem (< 5 bytes) 6 0 12(1/0/1) 12(1/1/1)
%bfins Mem (> 5 bytes) 6 0 18(2/0/2) 18(2/1/2)
bfffo Dn          20 0 20(0/0/0) 20(0/1/0)
%bfffo Mem (< 5 bytes) 6 0 22(1/0/0) 22(1/1/0)
%bfffo Mem (> 5 bytes) 6 0 28(2/0/0) 28(2/1/0)
bcc (taken)       6 0 6(0/0/0) 8(0/2/0)
bcc.b (not taken) 4 0 4(0/0/0) 4(0/1/0)
bcc.w (not taken) 6 0 6(0/0/0) 6(0/1/0)
bcc.l (not taken) 6 0 6(0/0/0) 8(0/2/0)
dbcc (false,cnt not expired) 6 0 6(0/0/0) 8(0/2/0)
dbcc (false,but cnt expired) 10 0 10(0/0/0) 13(0/3/0)
dbcc (true)       6 0 6(0/0/0) 8(0/1/0)
andi to sr        4 0 12(0/0/0) 14(0/2/0)
eori to sr        4 0 12(0/0/0) 14(0/2/0)
ori to sr         4 0 12(0/0/0) 14(0/2/0)
andi to ccr       4 0 12(0/0/0) 14(0/2/0)
eori to ccr       4 0 12(0/0/0) 14(0/2/0)

```

```

ori to ccr      4 0 12(0/0/0) 14(0/2/0)
bsr            2 0  6(0/0/1)  9(0/2/1)
%cas (succesful compare)  1 0 13(1/0/1) 13(1/1/1)
%cas (unsuccesful compare)  1 0 11(1/0/0) 11(1/1/0)
cas2 (succesful compare) (max)  2 0 24(2/0/2) 26(2/2/2)
cas2 (unsuccesful compare) (max)  2 0 24(2/0/0) 24(2/2/0)
chk Dn,Dn (no Exception)  8 0  8(0/0/0)  8(0/1/0)
chk Dn,Dn (Exception taken)  4 0 28(1/0/4) 30(1/3/4)
chk EA,Dn (no Exception)  0 0  8(0/0/0)  8(0/1/0)
chk EA,Dn (Exception taken)max  0 0 28(1/0/4) 30(1/3/4)
!chk2 Mem,Rn (no Exception)max  2 0 18(1/0/0) 18(1/1/0)
!chk2 Mem,Rn (Exception taken)mx2  0 40(2/0/4) 42(2/3/4)
jjmp          4 0  4(0/0/0)  6(0/2/0)
jjsr          0 0  4(0/0/1)  7(0/2/1)
#lea          2 0  2(0/0/0)  2(0/1/0)
link.w        0 0  4(0/0/1)  5(0/1/1)
link.l        2 0  6(0/0/1)  7(0/2/1)
nop           0 0  2(0/0/0)  2(0/1/0)
#pea          0 2  4(0/0/1)  4(0/1/1)
rtd           2 0 10(1/0/0) 12(1/2/0)
rtr           1 0 12(2/0/0) 14(2/2/0)
rts           1 0  9(1/0/0) 11(1/2/0)
unlk          0 0  5(1/0/0)  5(1/1/0)
bkpt          1 0  9(1/0/0)  9(1/0/0)
Interrupt (I-Stack)      0 0 23(2/0/4) 24(2/2/4)
Interrupt (M-Stack)      0 0 33(2/0/8) 34(2/2/8)
reset           0 0      518(0/0/0)      518(0/1/0)
stop           0 0  8(0/0/0)  8(0/2/0)
trace          0 0 22(1/0/5) 24(1/2/5)
trap #n        0 0 18(1/0/5) 20(1/2/4)
Illegal Instruction      0 0 18(1/0/5) 20(1/2/4)
A-Line trap           0 0 18(1/0/5) 20(1/2/4)
F-Line trap           0 0 18(1/0/5) 20(1/2/4)
Priviledge Violation    0 0 18(1/0/5) 20(1/2/4)
trapcc (Trap)         2 0 22(1/0/5) 24(1/2/5)
trapcc (No trap)      4 0  4(0/0/0)  4(0/1/0)
trapcc.w (Trap)       5 0 24(1/0/5) 26(1/3/5)
trapcc.w (No trap)    6 0  6(0/0/0)  6(0/1/0)
trapcc.l (Trap)       6 0 26(1/0/5) 28(1/3/5)
trapcc.l (No trap)    8 0  8(0/0/0)  8(0/2/0)
trapv (Trap)         2 0 22(1/0/5) 24(1/2/5)
trapv (No trap)      4 0  4(0/0/0)  4(0/1/0)

Bus Cycle Fault (Short)  0 0 36(1/0/10) 38(1/2/10)
Bus Cycle Fault (Long)  0 0 62(1/0/24) 64(1/2/24)
RTE (Normal Four Word)  1 0 18(4/0/0) 20(4/2/0)
RTE (Six Word)          1 0 18(4/0/0) 20(4/2/0)
RTE (Throwaway)         1 0 12(4/0/0) 12(4/0/0)
RTE (Coprocessor)       1 0 26(7/0/0) 26(7/2/0)
RTE (Short Fault)       1 0 36(10/0/0) 26(10/2/0)
RTE (Long Fault)        1 0 76(25/0/0) 76(25/2/0)

```

## 1.7 Fetch Effective Address (FEA)

	Head	Tail	I-Cache	No-Cache
Dn	- -	0 (0/0/0)	0 (0/0/0)	
An	- -	0 (0/0/0)	0 (0/0/0)	
(An)	1 1	3 (1/0/0)	3 (1/0/0)	
(An)+	0 1	3 (1/0/0)	3 (1/0/0)	
-(An)	2 2	4 (1/0/0)	4 (1/0/0)	
(d16,An)	2 2	4 (1/0/0)	4 (1/1/0)	
(d16,PC)	2 2	4 (1/0/0)	4 (1/1/0)	
(xxx).w	2 2	4 (1/0/0)	4 (1/1/0)	
(xxx).l	1 0	4 (1/0/0)	5 (1/1/0)	
#(data).b	2 0	2 (0/0/0)	2 (0/1/0)	
#(data).w	2 0	2 (0/0/0)	2 (0/1/0)	
#(data).l	4 0	4 (0/0/0)	4 (0/1/0)	
(d16,An)	2 0	6 (1/0/0)	7 (1/1/0)	
(d16,PC)	2 0	6 (1/0/0)	7 (1/1/0)	
(d16,An,Xn)	4 0	6 (1/0/0)	7 (1/1/0)	
(d16,PC,Xn)	4 0	6 (1/0/0)	7 (1/1/0)	
([d16,An])	2 0	10 (2/0/0)	10 (2/1/0)	
([d16,PC])	2 0	10 (2/0/0)	10 (2/1/0)	
([d16,An],Xn)	2 0	10 (2/0/0)	10 (2/1/0)	
([d16,PC],Xn)	2 0	10 (2/0/0)	10 (2/1/0)	
([d16,An],d16)	2 0	12 (2/0/0)	13 (2/2/0)	
([d16,PC],d16)	2 0	12 (2/0/0)	13 (2/2/0)	
([d16,An],Xn,d16)	2 0	12 (2/0/0)	13 (2/2/0)	
([d16,PC],Xn,d16)	2 0	12 (2/0/0)	13 (2/2/0)	
([d16,An],d32)	2 0	12 (2/0/0)	14 (2/2/0)	
([d16,PC],d32)	2 0	12 (2/0/0)	14 (2/2/0)	
([d16,An],Xn,d32)	2 0	12 (2/0/0)	14 (2/2/0)	
([d16,PC],Xn,d32)	2 0	12 (2/0/0)	14 (2/2/0)	
(B)	4 0	6 (1/0/0)	7 (1/1/0)	
(d16,B)	4 0	8 (1/0/0)	10 (1/1/0)	
(d32,B)	4 0	12 (1/0/0)	13 (1/2/0)	
([B])	4 0	10 (2/0/0)	10 (2/1/0)	
([B].l)	4 0	10 (2/0/0)	10 (2/1/0)	
([B],d16)	4 0	12 (2/0/0)	13 (2/1/0)	
([B].l,d16)	4 0	12 (2/0/0)	13 (2/1/0)	
([B],d32)	4 0	12 (2/0/0)	14 (2/2/0)	
([B].l,d32)	4 0	12 (2/0/0)	14 (2/2/0)	
([d16,B])	4 0	12 (2/0/0)	13 (2/1/0)	
([d16,B].l)	4 0	12 (2/0/0)	13 (2/1/0)	
([d16,B],d16)	4 0	14 (2/0/0)	16 (2/2/0)	
([d16,B].l,d16)	4 0	14 (2/0/0)	16 (2/2/0)	
([d16,B],d32)	4 0	14 (2/0/0)	17 (2/2/0)	
([d16,B].l,d32)	4 0	14 (2/0/0)	17 (2/2/0)	
([d32,B])	4 0	16 (2/0/0)	17 (2/2/0)	
([d32,B].l)	4 0	16 (2/0/0)	17 (2/2/0)	
([d32,B],d16)	4 0	18 (2/0/0)	20 (2/2/0)	
([d32,B].l,d16)	4 0	18 (2/0/0)	20 (2/2/0)	
([d32,B],d32)	4 0	18 (2/0/0)	21 (2/3/0)	
([d32,B].l,d32)	4 0	18 (2/0/0)	21 (2/3/0)	

## 1.8 Fetch Immediate Effective Address (FIEA)

Head	Tail	I-Cache	No-Cache
------	------	---------	----------

---

```

#(data).w,Dn      2+op  0  2(0/0/0)  2(0/1/0)
#(data).l,Dn      4+op  0  4(0/0/0)  4(0/1/0)
#(data).w,(An)    1  1  3(1/0/0)  4(1/1/0)
#(data).l,(An)    1  0  4(1/0/0)  5(1/1/0)
#(data).w,(An)+   2  1  5(1/0/0)  5(1/1/0)
#(data).l,(An)+   4  1  7(1/0/0)  7(1/1/0)
#(data).w,-(An)   2  2  4(1/0/0)  4(1/1/0)
#(data).l,-(An)   2  0  4(1/0/0)  5(1/1/0)
#(data).w,(d16,An) 2  0  4(1/0/0)  5(1/1/0)
#(data).l,(d16,An) 4  0  6(1/0/0)  8(1/2/0)
#(data).w,(xxx).w  4  2  6(1/0/0)  6(1/1/0)
#(data).l,(xxx).w  6  2  8(1/0/0)  8(1/2/0)
#(data).w,(xxx).l  3  0  6(1/0/0)  7(1/2/0)
#(data).l,(xxx).l  5  0  8(1/0/0)  9(1/2/0)
#(data).w,#(data).l 6+op  0  6(0/0/0)  6(0/2/0)
#(data).w,(d8,An,Xn) 6  2  8(1/0/0)  8(1/2/0)
#(data).w,(d8,PC,Xn) 6  2  8(1/0/0)  8(1/2/0)
#(data).l,(d8,An,Xn) 8  2 10(1/0/0) 10(1/2/0)
#(data).l,(d8,PC,Xn) 8  2 10(1/0/0) 10(1/2/0)
#(data).w,(d16,An) 4  0  8(1/0/0)  9(1/2/0)
#(data).w,(d16,PC) 4  0  8(1/0/0)  9(1/2/0)
#(data).l,(d16,An) 6  0 10(1/0/0) 11(1/2/0)
#(data).l,(d16,PC) 6  0 10(1/0/0) 11(1/2/0)
#(data).w,(d16,An,Xn) 6  0  8(1/0/0)  9(1/2/0)
#(data).w,(d16,PC,Xn) 6  0  8(1/0/0)  9(1/2/0)
#(data).l,(d16,An,Xn) 8  0 10(1/0/0) 11(1/2/0)
#(data).l,(d16,PC,Xn) 8  0 10(1/0/0) 11(1/2/0)
#(data).w,([d16,An]) 4  0 12(2/0/0) 14(2/2/0)
#(data).w,([d16,PC]) 4  0 12(2/0/0) 14(2/2/0)
#(data).l,([d16,An]) 6  0 14(2/0/0) 14(2/2/0)
#(data).l,([d16,PC]) 6  0 14(2/0/0) 14(2/2/0)
#(data).w,([d16,An],Xn) 4  0 12(2/0/0) 12(2/2/0)
#(data).w,([d16,PC],Xn) 4  0 12(2/0/0) 12(2/2/0)
#(data).l,([d16,An],Xn) 6  0 14(2/0/0) 14(2/2/0)
#(data).l,([d16,PC],Xn) 6  0 14(2/0/0) 14(2/2/0)
#(data).w,([d16,An],d16) 4  0 14(2/0/0) 15(2/2/0)
#(data).w,([d16,PC],d16) 4  0 14(2/0/0) 15(2/2/0)
#(data).l,([d16,An],d16) 6  0 16(2/0/0) 17(2/3/0)
#(data).l,([d16,PC],d16) 6  0 16(2/0/0) 17(2/3/0)
#(data).w,([d16,An],Xn,d16) 4  0 14(2/0/0) 15(2/2/0)
#(data).w,([d16,PC],Xn,d16) 4  0 14(2/0/0) 15(2/2/0)
#(data).l,([d16,An],Xn,d16) 6  0 16(2/0/0) 17(2/3/0)
#(data).l,([d16,PC],Xn,d16) 6  0 16(2/0/0) 17(2/3/0)
#(data).w,([d16,An],d32) 4  0 14(2/0/0) 16(2/3/0)
#(data).w,([d16,PC],d32) 4  0 14(2/0/0) 16(2/3/0)
#(data).l,([d16,An],d32) 6  0 16(2/0/0) 18(2/3/0)
#(data).l,([d16,PC],d32) 6  0 16(2/0/0) 18(2/3/0)
#(data).w,([d16,An],Xn,d32) 4  0 14(2/0/0) 16(2/3/0)
#(data).w,([d16,PC],Xn,d32) 4  0 14(2/0/0) 16(2/3/0)
#(data).l,([d16,An],Xn,d32) 6  0 16(2/0/0) 18(2/3/0)
#(data).l,([d16,PC],Xn,d32) 6  0 16(2/0/0) 18(2/3/0)
#(data).w,(B)      6  0  8(1/0/0)  9(1/1/0)
#(data).l,(B)      8  0 10(1/0/0) 11(1/2/0)
#(data).w,(d16,B)  6  0 10(1/0/0) 12(1/2/0)
#(data).l,(d16,B)  8  0 12(1/0/0) 14(1/2/0)
#(data).w,(d32,B) 10  0 14(1/0/0) 16(1/2/0)
#(data).l,(d32,B) 12  0 16(1/0/0) 18(1/3/0)

```

```

#(data).w, ([B])      6 0 12(2/0/0) 12(2/1/0)
#(data).l, ([B])      8 0 14(2/0/0) 14(2/2/0)
#(data).w, ([B].l)    6 0 12(2/0/0) 12(2/1/0)
#(data).l, ([B].l)    8 0 14(2/0/0) 14(2/2/0)
#(data).w, ([B],d16)  6 0 14(2/0/0) 15(2/2/0)
#(data).l, ([B],d16)  8 0 16(2/0/0) 17(2/2/0)
#(data).w, ([B].l,d16) 6 0 14(2/0/0) 15(2/2/0)
#(data).l, ([B].l,d16) 8 0 16(2/0/0) 17(2/2/0)
#(data).w, ([B],d32)  6 0 14(2/0/0) 16(2/2/0)
#(data).l, ([B],d32)  8 0 16(2/0/0) 18(2/3/0)
#(data).w, ([B].l,d32) 6 0 14(2/0/0) 16(2/2/0)
#(data).l, ([B].l,d32) 8 0 16(2/0/0) 18(2/3/0)
#(data).w, ([d16,B])  6 0 14(2/0/0) 15(2/2/0)
#(data).l, ([d16,B])  8 0 16(2/0/0) 17(2/2/0)
#(data).w, ([d16,B].l) 6 0 14(2/0/0) 15(2/2/0)
#(data).l, ([d16,B].l) 8 0 16(2/0/0) 17(2/2/0)
#(data).w, ([d16,B],d16) 6 0 16(2/0/0) 18(2/2/0)
#(data).l, ([d16,B],d16) 8 0 18(2/0/0) 20(2/3/0)
#(data).w, ([d16,B].l,d16) 6 0 16(2/0/0) 18(2/2/0)
#(data).l, ([d16,B].l,d16) 8 0 18(2/0/0) 20(2/3/0)
#(data).w, ([d16,B],d32) 6 0 16(2/0/0) 19(2/3/0)
#(data).l, ([d16,B],d32) 8 0 18(2/0/0) 21(2/3/0)
#(data).w, ([d16,B].l,d32) 6 0 16(2/0/0) 19(2/3/0)
#(data).l, ([d16,B].l,d32) 8 0 18(2/0/0) 21(2/3/0)
#(data).w, ([d32,B])  6 0 18(2/0/0) 19(2/2/0)
#(data).l, ([d32,B])  8 0 20(2/0/0) 21(2/3/0)
#(data).w, ([d32,B].l) 6 0 18(2/0/0) 19(2/2/0)
#(data).l, ([d32,B].l) 8 0 20(2/0/0) 21(2/3/0)
#(data).w, ([d32,B],d16) 6 0 20(2/0/0) 22(2/3/0)
#(data).l, ([d32,B],d16) 8 0 22(2/0/0) 24(2/3/0)
#(data).w, ([d32,B].l,d16) 6 0 20(2/0/0) 22(2/3/0)
#(data).l, ([d32,B].l,d16) 8 0 22(2/0/0) 24(2/3/0)
#(data).w, ([d32,B],d32) 6 0 20(2/0/0) 23(2/3/0)
#(data).l, ([d32,B],d32) 8 0 22(2/0/0) 25(2/4/0)
#(data).w, ([d32,B].l,d32) 6 0 20(2/0/0) 23(2/3/0)
#(data).l, ([d32,B].l,d32) 8 0 22(2/0/0) 25(2/4/0)

```

## 1.9 Calculate Effective Address (CEA)

	Head	Tail	I-Cache	No-Cache
Dn	- -	0(0/0/0)	0(0/0/0)	
An	- -	0(0/0/0)	0(0/0/0)	
(An)	2+op	0	2(0/0/0)	2(0/0/0)
(An)+	0 0	2(0/0/0)	2(0/0/0)	
-(An)	2+op	0	2(0/0/0)	2(0/0/0)
(d16,An)	2+op	0	2(0/0/0)	2(0/0/0)
(d16,PC)	2+op	0	2(0/0/0)	2(0/1/0)
(xxx).w	2+op	0	2(0/0/0)	2(0/1/0)
(xxx).l	4+op	0	4(0/0/0)	4(0/1/0)
(d8,An,Xn)	4+op	0	4(0/0/0)	4(0/1/0)
(d8,PC,Xn)	4+op	0	4(0/0/0)	4(0/1/0)
(d16,An)	2 0	6(0/0/0)	6(0/1/0)	
(d16,PC)	2 0	6(0/0/0)	6(0/1/0)	
(d16,An,Xn)	6+op	0	6(0/0/0)	6(0/1/0)
(d16,PC,Xn)	6+op	0	6(0/0/0)	6(0/1/0)

```

([d16,An])      2 0 10(1/0/0) 10(1/1/0)
([d16,PC])      2 0 10(1/0/0) 10(1/1/0)
([d16,An],Xn)   2 0 10(1/0/0) 10(1/1/0)
([d16,PC],Xn)   2 0 10(1/0/0) 10(1/1/0)
([d16,An],d16)  2 0 12(1/0/0) 13(1/2/0)
([d16,PC],d16)  2 0 12(1/0/0) 13(1/2/0)
([d16,An],Xn,d16) 2 0 12(1/0/0) 13(1/2/0)
([d16,PC],Xn,d16) 2 0 12(1/0/0) 13(1/2/0)
([d16,An],d32)  2 0 12(1/0/0) 13(1/2/0)
([d16,PC],d32)  2 0 12(1/0/0) 13(1/2/0)
([d16,An],Xn,d32) 2 0 12(1/0/0) 11(1/2/0)
([d16,PC],Xn,d32) 2 0 12(1/0/0) 13(1/2/0)
(B)             6+op 0 6(0/0/0) 6(0/1/0)
(d16,B)         4 0 8(0/0/0) 9(0/1/0)
(d32,B)         4 0 12(0/0/0) 12(0/2/0)
([B])           4 0 10(1/0/0) 10(1/1/0)
([B].l)         4 0 10(1/0/0) 10(1/1/0)
([B],d16)       4 0 12(1/0/0) 13(1/1/0)
([B].l,d16)     4 0 12(1/0/0) 13(1/1/0)
([B],d32)       4 0 12(1/0/0) 13(1/2/0)
([B].l,d32)     4 0 12(2/0/0) 13(1/2/0)
([d16,B])       4 0 12(1/0/0) 13(1/1/0)
([d16,B].l)     4 0 12(1/0/0) 13(1/1/0)
([d16,B],d16)   4 0 14(1/0/0) 16(1/2/0)
([d16,B].l,d16) 4 0 14(1/0/0) 16(1/2/0)
([d16,B],d32)   4 0 14(1/0/0) 16(1/2/0)
([d16,B].l,d32) 4 0 14(1/0/0) 16(1/2/0)
([d32,B])       4 0 16(1/0/0) 17(1/2/0)
([d32,B].l)     4 0 16(1/0/0) 17(1/2/0)
([d32,B],d16)   4 0 18(1/0/0) 20(1/2/0)
([d32,B].l,d16) 4 0 18(1/0/0) 20(1/2/0)
([d32,B],d32)   4 0 18(1/0/0) 20(1/3/0)
([d32,B].l,d32) 4 0 18(1/0/0) 20(1/3/0)

```

## 1.10 Calculate Immediate Effective Address (CIEA)

Calculate Immediate Effective Address (CIEA) for WORDS  
(for LONGS add 2 for head and cycle count)

	Head	Tail	I-Cache	No-Cache
Dn	2+op 0	2(0/0/0)	0(0/0/0)	
(An)	2 0	2(0/0/0)	2(0/0/0)	
(An)+	2+op 0	4(0/0/0)	2(0/0/0)	
-(An)	2+op 0	2(0/0/0)	2(0/0/0)	
(d16,An)	4+op 0	4(0/0/0)	2(0/0/0)	
(d16,PC)	4+op 0	4(0/0/0)	2(0/1/0)	
(xxx).w	4+op 0	4(0/0/0)	2(0/1/0)	
(xxx).l	6+op 0	6(0/0/0)	4(0/1/0)	
(d8,An,Xn)	6+op 0	6(0/0/0)	4(0/1/0)	
(d8,PC,Xn)	6+op 0	6(0/0/0)	4(0/1/0)	
(d16,An)	4 0	8(0/0/0)	6(0/1/0)	
(d16,PC)	4 0	8(0/0/0)	6(0/1/0)	
(d16,An,Xn)	4+op 0	8(0/0/0)	6(0/1/0)	
(d16,PC,Xn)	4+op 0	8(0/0/0)	6(0/1/0)	
([d16,An])	4 0	12(1/0/0)	6(1/1/0)	

```

([d16,PC])      4 0 12(1/0/0)  6(1/1/0)
([d16,An],Xn)   8+op 0 10(2/0/0) 10(2/1/0)
([d16,PC],Xn)   8+op 0 10(2/0/0) 10(2/1/0)
([d16,An],d16)  4 0 12(2/0/0) 13(2/2/0)
([d16,PC],d16)  4 0 12(2/0/0) 13(2/2/0)
([d16,An],Xn,d16) 4 0 12(2/0/0) 13(2/2/0)
([d16,PC],Xn,d16) 4 0 12(2/0/0) 13(2/2/0)
([d16,An],d32)  4 0 12(2/0/0) 14(2/2/0)
([d16,PC],d32)  4 0 12(2/0/0) 14(2/2/0)
([d16,An],Xn,d32) 4 0 12(2/0/0) 14(2/2/0)
([d16,PC],Xn,d32) 4 0 12(2/0/0) 14(2/2/0)
(B)             8+op 0 6(1/0/0)  7(1/1/0)
(d16,B)         6 0 8(1/0/0) 10(1/1/0)
(d32,B)         6 0 12(1/0/0) 13(1/2/0)
([B])           6 0 12(1/0/0) 12(1/1/0)
([B].l)         6 0 12(1/0/0) 12(1/1/0)
([B],d16)       6 0 14(1/0/0) 15(1/2/0)
([B].l,d16)     6 0 14(1/0/0) 15(1/2/0)
([B],d32)       6 0 14(1/0/0) 15(1/2/0)
([B].l,d32)     6 0 14(1/0/0) 15(1/2/0)
([d16,B])       6 0 14(1/0/0) 15(1/2/0)
([d16,B].l)     6 0 14(1/0/0) 15(1/2/0)
([d16,B],d16)   6 0 16(1/0/0) 18(1/2/0)
([d16,B].l,d16) 6 0 16(1/0/0) 18(1/2/0)
([d16,B],d32)   6 0 16(1/0/0) 18(1/2/0)
([d16,B].l,d32) 6 0 16(1/0/0) 18(1/2/0)
([d32,B])       6 0 18(1/0/0) 19(1/2/0)
([d32,B].l)     6 0 18(2/0/0) 19(2/2/0)
([d32,B],d16)   6 0 20(1/0/0) 22(1/3/0)
([d32,B].l,d16) 6 0 20(1/0/0) 22(1/3/0)
([d32,B],d32)   6 0 22(1/0/0) 24(1/3/0)
([d32,B].l,d32) 6 0 22(1/0/0) 24(1/3/0)

```

## 1.11 Jump Effective Address (JEA)

	Head	Tail	I-Cache	No-Cache
(An)	2+op 0	2(0/0/0)	2(0/0/0)	
(d16,An)	4+op 0	4(0/0/0)	4(0/0/0)	
\$00.w	2+op 0	2(0/0/0)	2(0/0/0)	
\$00.l	2+op 0	2(0/0/0)	2(0/0/0)	
(d8,An,Xn)	6+op 0	6(0/0/0)	6(0/0/0)	
(d8,An,PC)	6+op 0	6(0/0/0)	6(0/0/0)	
(d16,An)	2 0	6(0/0/0)	6(0/0/0)	
(d16,PC)	2 0	6(0/0/0)	6(0/0/0)	
(d16,An,Xn)	6+op 0	6(0/0/0)	6(0/0/0)	
(d16,PC,Xn)	6+op 0	6(0/0/0)	6(0/0/0)	
([d16,An])	2 0	10(1/0/0)	10(1/1/0)	
([d16,PC])	2 0	10(1/0/0)	10(1/1/0)	
([d16,An],Xn)	2 0	10(1/0/0)	10(1/1/0)	
([d16,PC],Xn)	2 0	10(1/0/0)	10(1/1/0)	
([d16,An],d16)	2 0	12(1/0/0)	12(1/2/0)	
([d16,PC],d16)	2 0	12(1/0/0)	12(1/2/0)	
([d16,An],Xn,d16)	2 0	12(1/0/0)	12(1/2/0)	
([d16,PC],Xn,d16)	2 0	12(1/0/0)	12(1/2/0)	
([d16,An],d32)	2 0	12(1/0/0)	12(1/2/0)	

```

([d16,PC],d32)      2 0 12(1/0/0) 12(1/2/0)
([d16,An],Xn,d32)  2 0 12(1/0/0) 12(1/2/0)
([d16,PC],Xn,d32)  2 0 12(1/0/0) 12(1/2/0)
(B)                6+op 0 6(0/0/0) 6(0/0/0)
(d16,B)            4 0 8(0/0/0) 9(0/1/0)
(d32,B)            4 0 12(0/0/0) 13(0/1/0)
([B])              4 0 10(1/0/0) 10(1/1/0)
([B].l)            4 0 10(1/0/0) 10(1/1/0)
([B],d16)          4 0 12(1/0/0) 12(1/1/0)
([B].l,d16)        4 0 12(1/0/0) 12(1/1/0)
([B],d32)          4 0 12(1/0/0) 12(1/1/0)
([B].l,d32)        4 0 12(1/0/0) 12(1/1/0)
([d16,B])          4 0 12(1/0/0) 13(1/1/0)
([d16,B].l)        4 0 12(1/0/0) 13(1/1/0)
([d16,B],d16)      4 0 14(1/0/0) 15(1/1/0)
([d16,B].l,d16)    4 0 14(1/0/0) 15(1/1/0)
([d16,B],d32)      4 0 14(1/0/0) 15(1/1/0)
([d16,B].l,d32)    4 0 14(1/0/0) 15(1/1/0)
([d32,B])          4 0 16(1/0/0) 17(1/2/0)
([d32,B].l)        4 0 16(1/0/0) 17(1/2/0)
([d32,B],d16)      4 0 18(1/0/0) 19(1/2/0)
([d32,B].l,d16)    4 0 18(1/0/0) 19(1/2/0)
([d32,B],d32)      4 0 18(1/0/0) 19(1/2/0)
([d32,B].l,d32)    4 0 18(1/0/0) 19(1/2/0)

```

## 1.12 Some notes about 020+

Most of 020 cycletimes are same than on 030; haven't found any differenties.

```

;-----
...
clr.l -(a0)
clr.l -(a0)
...
is 25% faster than
...
clr.l (a0)+
clr.l (a0)+
...
;-----
move.l (a0,d0.l)  is faster than
move.l (a0,d0.w)
same with adda.w <-> adda.l etc.
;-----
move.b d0,-(a7) will decrease a7 with 2!
;-----

```

Never use those silly #xx,([a0]) new modes, they are slower than  
 move.l (a0),a0  
 move.l #xx,(a0)  
 ...case you have no extra spare registers, but... that's rare.

```

;-----

Time-optimizing on A1200 is more sparing BUS than CPU; the CHIP RAM

```

is just too slow.. Remember to adjust your WRITES to CHIP; Adjusting memory READS wont do you no good. (Naturally, how could computer keep going on without knowing have we read something we're using?)

;-----

Interrupts are deadly slow, try to figure out something better.. like Copper ;-). Just plain `movem*2`, Trap and RTE will take HUGE amount of cycles.

```
movem.l all,-(sp)
movem.l (sp)+,all
rte
```

will take about 80 Cycles... okey, why use all regs ;-)

## 1.13 Arithmetics Instructions

- ADD
- ADDA
- ADDI
- ADDQ
- ADDX
- SUB
- SUBA
- SUBI
- SUBQ
- SUBX
- DIVS
- DIVU
- MULS
- MULU
- CLR
- CMP
- CMPA
- CMPI
- CMPM
- CMP2
- EXT
- NEG
- NEGX
- NOF

## 1.14 Logic Instructions

- AND
- ANDI
- EOR
- EORI
- OR
- ORI
- NOT

---

TST  
SCC

## 1.15 Shift and Rotate Instructions

ASL  
ASR  
LSL  
LSR  
ROL  
ROR  
ROXL  
ROXR  
SWAP

## 1.16 Bit Manipulation Instructions

BCHG  
BCLR  
BSET  
BTST

## 1.17 Bit Field Manipulation Instructions

BFCHG  
BFCLR  
BFSET  
BFST  
BFEXTS  
BFEXTU  
BFFFO  
BFINS

## 1.18 BCD Instructions

ABCD  
NBCD  
SB CD  
PACK  
UNPK

## 1.19 Data Transfer Instructions

---

EXG  
LEA  
LINK  
MOVE  
MOVEA  
MOVEM  
MOVEQ  
MOVEP  
PEA  
UNLK

## 1.20 Flow Control Instructions

Bcc  
DBcc  
BRA  
BSR  
JMP  
JSR  
RTD  
RTR  
RTS

## 1.21 Privileged Instructions

ANDI SR  
EORI SR  
ORI SR  
MOVE to SR  
MOVE from SR  
MOVE USP  
MOVEC  
MOVES  
RESET  
RTE  
STOP

## 1.22 'Exceptions' Instructions

BKPT  
CHK  
CHK2  
ILLEGAL  
TRAP  
TRAPcc  
TRAPV

---

## 1.23 CCR related Instructions

ANDI CCR  
EORI CCR  
ORI CCR  
MOVE from CCR  
MOVE to CCR

## 1.24 PMMU control Instructions

PFLUSH  
PFLUSHA  
PLOADR  
PLOADW  
PMOVE  
PTESTR  
PTESTW  
CALLM  
RTM

## 1.25 Multiprocessor Instructions

TAS  
CAS  
CAS2

## 1.26 CoProcessor Instructions

cpBcc  
cpDBcc  
cpGEN  
cpRESTORE  
cpSAVE  
cpScc  
cpTRAPcc

## 1.27 Alphabetical Index

ABCD  
ADD  
ADDA  
ADDI  
ADDQ  
ADDX  
AND  
ANDI  
ANDI CCR

---

---

ANDI SR  
ASL  
ASR  
Bcc  
BCHG  
BCLR  
BFCHG  
BFCLR  
BFEXTS  
BFEXTU  
BFFFO  
BFINS  
BFSET  
BFTST  
BKPT  
BRA  
BSET  
BSR  
BTST  
CALLM  
CAS  
CAS2  
CHK  
CHK2  
CLR  
CMP  
CMP2  
CMPA  
CMPI  
CMPM  
cpBcc  
cpDBcc  
cpGEN  
cpRESTORE  
cpSAVE  
cpScc  
cpTRAPcc  
DBcc  
DIVS  
DIVU  
EOR  
EORI  
EORI CCR  
EORI SR  
EXG  
EXT  
ILLEGAL  
JMP  
JSR  
LEA  
LINK  
LSL  
LSR  
MOVE  
MOVEA  
MOVE from CCR  
MOVE to CCR

---

---

MOVE from SR  
MOVE to SR  
MOVE from/to USP  
MOVEC  
MOVEM  
MOVEP  
MOVEQ  
MOVES  
MUL  
NBCD  
NEG  
NEGX  
NOP  
NOT  
OR  
ORI  
ORI CCR  
ORI SR  
PACK  
PEA  
PFLUSH  
PFLUSHA  
PLOADR  
PLOADW  
PMOVE  
PTESTR  
PTESTW  
RESET  
ROL  
ROR  
ROXL  
ROXR  
RTD  
RTE  
RTM  
RTR  
RTS  
SBCD  
Scc  
STOP  
SUB  
SUBA  
SUBI  
SUBQ  
SUBX  
SWAP  
TAS  
TRAP  
TRAP<sub>CC</sub>  
TRAPV  
TST  
UNLK  
UNPK

---

## 1.28 Add Binary Coded Decimal (w/extend)

### NAME

ABCD -- Add binary coded decimal

### SYNOPSIS

ABCD Dy,Dx  
ABCD -(Ay),-(Ax)

Size = (Byte)

### FUNCTION

Adds the source operand to the destination operand along with the extend bit, and stores the result in the destination location. The addition is performed using binary coded decimal arithmetic. The operands, which are packed BCD numbers, can be addressed in two different ways:

1. Data register to data register: The operands are contained in the data registers specified in the instruction.
2. Memory to memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

This operation is a byte operation only.

Normally the Z condition code bit is set via programming before the start of an operation. That allows successful tests for zero results upon completion of multiple-precision operations.

### FORMAT

```
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|-----|---|---|---|---|---|---|-----|
| 1| 1| 0| 0|   Rx   | 1| 0| 0| 0| 0|R/M|   Ry   |
-----
```

R/M = 0 -> data register  
R/M = 1 -> address register  
Rx: destination register  
Ry: source register

### RESULT

X - Set the same as the carry bit.  
N - Undefined  
Z - Cleared if the result is non-zero. Unchanged otherwise.  
V - Undefined  
C - Set if a decimal carry was generated. Cleared otherwise.

### SEE ALSO

ADD ADDI ADDQ  
ADDX  
SUB SUBI SUBQ  
SBCD SUBX

## 1.29 ADD integer

### NAME

ADD -- Add integer

### SYNOPSIS

ADD <ea>,Dn

ADD Dn,<ea>

Size = (Byte, Word, Long)

### FUNCTION

Adds the source operand to the destination operand using binary addition, and stores the result in the destination location. The size of the operation may be specified as byte, word, or long. The mode of the instruction indicates which operand is the source and which is the destination as well as the operand size.

### FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|-----|-----|-----|-----|
| 1 | 1 | 0 | 1 | REGISTER | OP-MODE | MODE | REGISTER |
=====
<ea>

```

### OP-MODE

Byte	Word	Long	
000	001	010	(Dn) + (<ea>) -> Dn
100	101	110	(<ea>) + (Dn) -> <ea>

### REGISTER

One of the 8 datas registers

If <ea> is source, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An *	001	N\textdegree{} reg. An	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	111	011
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	111	100
([bd,An,Xi]od)	110	N\textdegree{} reg. An			

```
|-----|
| ([bd,An],Xi,od) |110 |N\textdegree{} reg. An|
|-----|
```

\* Word or Long only

If <ea> is destination, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111   000	
An	-	-	Abs.L	111   001	
(An)	010	N\textdegree{} reg. An	(d16,PC)	-   -	
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-   -	
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-   -	
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-   -	
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-   -	
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-   -	
-----					
([bd,An,Xi]od)  110  N\textdegree{} reg. An					
-----					
([bd,An],Xi,od)  110  N\textdegree{} reg. An					
-----					

When destination is an Address Register, ADDA instruction is used.

#### RESULT

X - Set the same as the carry bit.  
 N - Set if the result is negative. Cleared otherwise.  
 Z - Set if the result is zero. Cleared otherwise.  
 V - Set if an overflow is generated. Cleared otherwise.  
 C - Set if a carry is generated. Cleared otherwise.

#### SEE ALSO

ADDI ADDQ ADDX  
 SUB SUBI SUBQ  
 SUBX

## 1.30 ADD Address

#### NAME

ADDA -- Add address

#### SYNOPSIS

ADDA <ea>,An

Size = (Word, Long)

#### FUNCTION

Adds the source operand to the destination address register,

and stores the result in the destination address register. The size of the operation may be specified as word or long. The entire destination operand is used regardless of the operation size.

#### FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|-----|-----|-----|-----|
| 1 | 1 | 0 | 1 | REGISTER | OP-MODE | MODE | REGISTER |
-----=====

```

<ea>

#### OP-MODE

Indicates operation length:

011->one Word operation: source operand is extended to 32 bits

111->one Long operation

#### REGISTER

One of the 8 address registers.

<ea> is always source, all addressing modes are allowed.

```

-----
|Addressing Mode|Mode| Register | |Addressing Mode|Mode|Register|
|-----|-----|-----| |-----|-----|-----|
|      Dn      |000 |N\textdegree{} reg. Dn| |   Abs.W   |111 | 000 |
|-----|-----|-----| |-----|-----|-----|
|      An      |001 |N\textdegree{} reg. An| |   Abs.L   |111 | 001 |
|-----|-----|-----| |-----|-----|-----|
|     (An)     |010 |N\textdegree{} reg. An| | (d16,PC) |111 | 010 |
|-----|-----|-----| |-----|-----|-----|
|     (An)+    |011 |N\textdegree{} reg. An| | (d8,PC,Xi)|111 | 011 |
|-----|-----|-----| |-----|-----|-----|
|    -(An)    |100 |N\textdegree{} reg. An| | (bd,PC,Xi)|111 | 011 |
|-----|-----|-----| |-----|-----|-----|
| (d16,An)    |101 |N\textdegree{} reg. An| | ([bd,PC,Xi],od)|111 | 011 |
|-----|-----|-----| |-----|-----|-----|
| (d8,An,Xi)  |110 |N\textdegree{} reg. An| | ([bd,PC],Xi,od)|111 | 011 |
|-----|-----|-----| |-----|-----|-----|
| (bd,An,Xi)  |110 |N\textdegree{} reg. An| |   #data   |111 | 100 |
|-----|-----|-----| |-----|-----|-----|
| ([bd,An,Xi]od) |110 |N\textdegree{} reg. An|
|-----|-----|-----|
| ([bd,An],Xi,od) |110 |N\textdegree{} reg. An|
-----

```

#### RESULT

None.

#### SEE ALSO

ADDQ SUBQ SUBA

## 1.31 ADD Immediate

#### NAME

ADDI -- Add immediate

SYNOPSIS

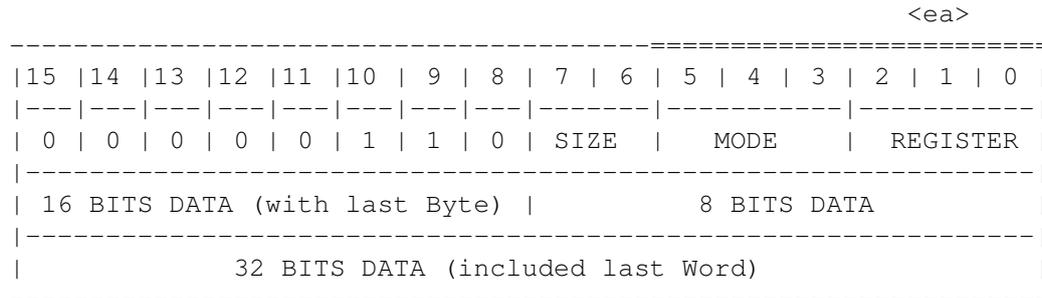
ADDI #<data>,<ea>

Size = (Byte, Word, Long)

FUNCTION

Adds the immediate data to the destination operand, and stores the result in the destination location. The size of the operation may be specified as byte, word, or long. The size of the immediate data matches the operation size.

FORMAT



SIZE

- 00->one Byte operation
- 01->one Word operation
- 10->one Long operation

REGISTER

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N <sup>th</sup> reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N <sup>th</sup> reg. An	(d16,PC)	-	-
(An)+	011	N <sup>th</sup> reg. An	(d8,PC,Xi)	-	-
-(An)	100	N <sup>th</sup> reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N <sup>th</sup> reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N <sup>th</sup> reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N <sup>th</sup> reg. An	#data	-	-
([bd,An,Xi]od)	110	N <sup>th</sup> reg. An			
([bd,An],Xi,od)	110	N <sup>th</sup> reg. An			

RESULT

X - Set the same as the carry bit.

N - Set if the result is negative. Cleared otherwise.  
 Z - Set if the result is zero. Cleared otherwise.  
 V - Set if an overflow is generated. Cleared otherwise.  
 C - Set if a carry is generated. Cleared otherwise.

SEE ALSO

ADD ADDQ ADDX  
 SUB SUBI SUBQ  
 SUBX

### 1.32 ADD 3-bit immediate Quick

NAME

ADDQ -- Add 3-bit immediate quick

SYNOPSIS

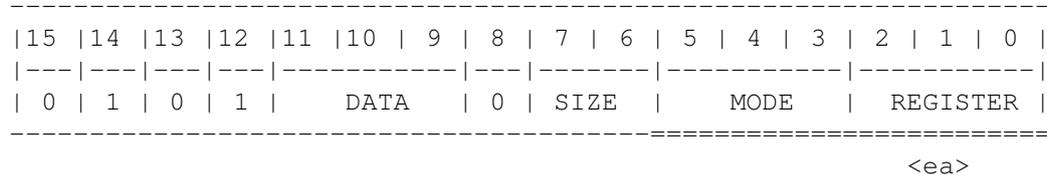
ADDQ #<data>, <ea>

Size = (Byte, Word, Long)

FUNCTION

Adds the immediate value of 1 to 8 to the operand at the destination location. The size of the operation may be specified as byte, word, or long. When adding to address registers, the condition codes are not altered, and the entire destination address register is used regardless of the operation size.

FORMAT



DATA

000 ->represent value 8  
 001 to 111 ->immediate data from 1 to 7

SIZE

00->one Byte operation  
 01->one Word operation  
 10->one Long operation

REGISTER

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N <sup>th</sup> reg. Dn	Abs.W	111	000
An *	001	N <sup>th</sup> reg. An	Abs.L	111	001
(An)	010	N <sup>th</sup> reg. An	(d16,PC)	-	-

(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-	
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-	
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-	
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-	
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-	
([bd,An,Xi]od)	110	N\textdegree{} reg. An				
([bd,An],Xi,od)	110	N\textdegree{} reg. An				

\* Word or Long only.

#### RESULT

- X - Set the same as the carry bit.
- N - Set if the result is negative. Cleared otherwise.
- Z - Set if the result is zero. Cleared otherwise.
- V - Set if an overflow is generated. Cleared otherwise.
- C - Set if a carry is generated. Cleared otherwise.

#### SEE ALSO

ADD ADDI  
SUB SUBI SUBQ

## 1.33 ADD integer with eXtend

#### NAME

ADDX -- Add integer with extend

#### SYNOPSIS

ADDX Dy,Dx  
ADDX -(Ay),-(Ax)

Size = (Byte, Word, Long)

#### FUNCTION

Adds the source operand to the destination operand along with the extend bit, and stores the result in the destination location. The addition is performed using binary coded decimal arithmetic. The operands, which are packed BCD numbers, can be addressed in two different ways:

1. Data register to data register: The operands are contained in the data registers specified in the instruction.
2. Memory to memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

The size of operation can be specified as byte, word, or long.

Normally the Z condition code bit is set via programming before the start of an operation. That allows successful tests for zero results upon completion of multiple-precision operations.

#### FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|-----|---|-----|---|---|---|-----|
| 1 | 1 | 0 | 1 |   Rx   | 1 | SIZE | 0 | 0 |R/M|   Ry   |
-----

```

R/M = 0 -> data register  
R/M = 1 -> address register  
Rx: destination register  
Ry: source register

#### SIZE

00->one Byte operation  
01->one Word operation  
10->one Long operation

#### RESULT

X - Set the same as the carry bit.  
N - Set if the result is negative. Cleared otherwise.  
Z - Cleared if the result is non-zero. Unchanged otherwise.  
V - Set if an overflow is generated. Cleared otherwise.  
C - Set if a carry is generated. Cleared otherwise.

#### SEE ALSO

ADD ADDI  
SUB SUBI SUBX

## 1.34 Logical AND

#### NAME

AND -- Logical AND

#### SYNOPSIS

AND <ea>,Dn  
AND Dn,<ea>

Size = (Byte, Word, Long)

#### FUNCTION

Performs a bit-wise AND operation with the source operand and the destination operand and stores the result in the destination. The size of the operation can be specified as byte, word, or long. The contents of an address register may not be used as an operand.

#### FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|-----|---|-----|---|---|---|-----|
| 1 | 1 | 0 | 0 | REGISTER | OP-MODE |   MODE   | REGISTER |
-----

```

&lt;ea&gt;

## OP-MODE

```

Byte  Word  Long
000 001 010 (Dn)AND(<ea>)-> Dn
100 101 110 (<ea>)AND(Dn)-> <ea>

```

## REGISTER

One of the 8 datas registers  
 If <ea> is source, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	111	011
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	111	100
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

If <ea> is destination, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-

```
| ([bd,An,Xi,od) |110 |N\textdegree{} reg. An|
|-----|
| ([bd,An],Xi,od) |110 |N\textdegree{} reg. An|
|-----|
```

AND between two data registers is allowed if you consider the syntax where Dn is at destination's place.

If you use this instruction with an immediate data, it does the same as instruction ANDI.

#### RESULT

X - Not affected  
 N - Set if the most-significant bit of the result was set. Cleared otherwise.  
 Z - Set if the result was zero. Cleared otherwise.  
 V - Always cleared.  
 C - Always cleared.

#### SEE ALSO

ANDI

## 1.35 Logical AND Immediate

#### NAME

ANDI -- Logical AND immediate

#### SYNOPSIS

ANDI #<data>, <ea>

Size = (Byte, Word, Long)

#### FUNCTION

Performs a bit-wise AND operation with the immediate data and the destination operand and stores the result in the destination. The size of the operation can be specified as byte, word, or long. The size of the immediate data matches the operation size.

#### FORMAT

	<ea>
-----	
15   14   13   12   11   10   9   8   7   6   5   4   3   2   1   0	
---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---	
0   0   0   0   0   0   1   0   SIZE   MODE   REGISTER	
-----	
16 BITS DATA (with last Byte)   8 BITS DATA	
-----	
32 BITS DATA (included last Word)	
-----	

#### SIZE

00->one Byte operation  
 01->one Word operation  
 10->one Long operation

## REGISTER

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111   001	
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

## RESULT

X - Not affected

N - Set if the most-significant bit of the result was set. Cleared otherwise.

Z - Set if the result was zero. Cleared otherwise.

V - Always cleared.

C - Always cleared.

## SEE ALSO

AND ANDI to CCR ANDI to SR

## 1.36 Logical AND Immediate to CCR

## NAME

ANDI to CCR -- Logical AND immediate to condition code register

## SYNOPSIS

ANDI #<data>,CCR

Size = (Byte)

## FUNCTION

Performs a bit-wise AND operation with the immediate data and the lower byte of the status register.

## FORMAT

|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

```

|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |           8 BITS IMMEDIATE DATA   |
-----

```

**RESULT**

X - Cleared if bit 4 of immed. operand is zero. Unchanged otherwise.  
N - Cleared if bit 3 of immed. operand is zero. Unchanged otherwise.  
Z - Cleared if bit 2 of immed. operand is zero. Unchanged otherwise.  
V - Cleared if bit 1 of immed. operand is zero. Unchanged otherwise.  
C - Cleared if bit 0 of immed. operand is zero. Unchanged otherwise.

**SEE ALSO**

AND ANDI ANDI to SR

**1.37 Logical AND Immediate to SR (privileged)****NAME**

ANDI to SR -- Logical AND immediate to status register (privileged)

**SYNOPSIS**

ANDI #<data>,SR

Size = (Word)

**FUNCTION**

Performs a bit-wise AND operation with the immediate data and the status register. All implemented bits of the status register are affected.

**FORMAT**

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
|-----|-----|-----|-----|-----|-----|-----|-----|
|                                     16 BITS IMMEDIATE DATA   |
-----

```

**RESULT**

X - Cleared if bit 4 of immed. operand is zero. Unchanged otherwise.  
N - Cleared if bit 3 of immed. operand is zero. Unchanged otherwise.  
Z - Cleared if bit 2 of immed. operand is zero. Unchanged otherwise.  
V - Cleared if bit 1 of immed. operand is zero. Unchanged otherwise.  
C - Cleared if bit 0 of immed. operand is zero. Unchanged otherwise.

**SEE ALSO**

AND ANDI ANDI to CCR

**1.38 Arithmetic Shift Left and Arithmetic Shift Right**

NAME

ASL, ASR -- Arithmetic shift left and arithmetic shift right

SYNOPSIS

ASd Dx,Dy

ASd #<data>,Dy

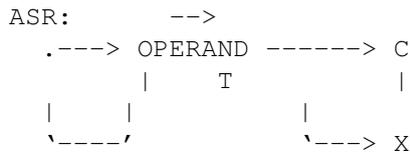
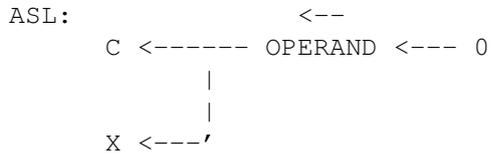
ASd <ea>

where d is direction, L or R

Size = (Byte, Word, Long)

FUNCTION

Performs an arithmetic shifting bit operation in the indicated direction, with an immediate data, or with a data register. If you shift address contents, you only can do ONE shift, and your operand is ONE word exclusively.



FORMAT

In the case of the shifting of a register:

```

~~~~~
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1| 1| 1| 0| NUMBER/ |dr| SIZE |i/r| 0| 0| REGISTER |
|  |  |  |  | REGISTER |  |      |  |  |  |  |
-----
```

In the case of the shifting of a memory area:

```

~~~~~
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1| 1| 1| 0| 0| 0| 0|dr| 1| 1|  | MODE  | REGISTER |
-----
```

<ea>

NUMBER/REGISTER

Specifies number of shifting or number of register which contents the number of shifting.

If i/r = 0, number of shifting is specified in the instruction as immediate data

If i/r = 1, it's specified in the data register.

If dr = 0, right shifting  
 If dr = 1, left shifting

#### SIZE

00->one Byte operation  
 01->one Word operation  
 10->one Long operation

#### REGISTER

For a register shifting:  
 Indicates the number of data register on which shifting is applied.

For a memory shifting:  
 <ea> indicates operand which should be shifted.  
 Only addressing modes relatives to memory are allowed:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

#### RESULT

X - Set according to the list bit shifted out of the operand.  
 Unaffected for a shift count of zero.  
 N - Set if the most-significant bit of the result is set. Cleared otherwise.  
 Z - Set if the result is zero. Cleared otherwise.  
 V - Set if the most significant bit is changed at any time during the shift operation. Cleared otherwise.  
 C - Set according to the list bit shifted out of the operand.  
 Cleared for a shift count of zero.

#### SEE ALSO

ROd ROXd



Size = (Byte, Long)

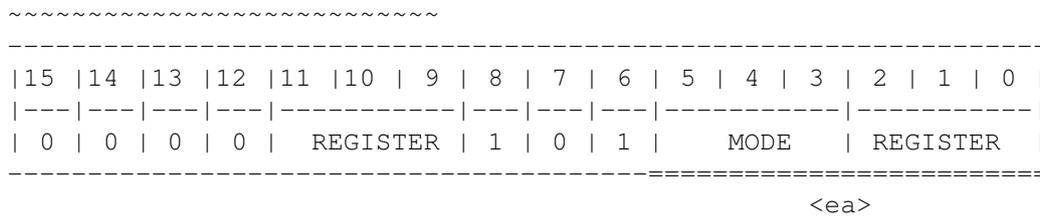
FUNCTION

Tests a bit in the destination operand and sets the Z condition code appropriately, then inverts the bit in the destination. If the destination is a data register, any of the 32 bits can be specified by the modulo 32 number. When the destination is a memory location, the operation must be a byte operation, and therefore the bit number is modulo 8. In all cases, bit zero is the least significant bit. The bit number for this operation may be specified in either of two ways:

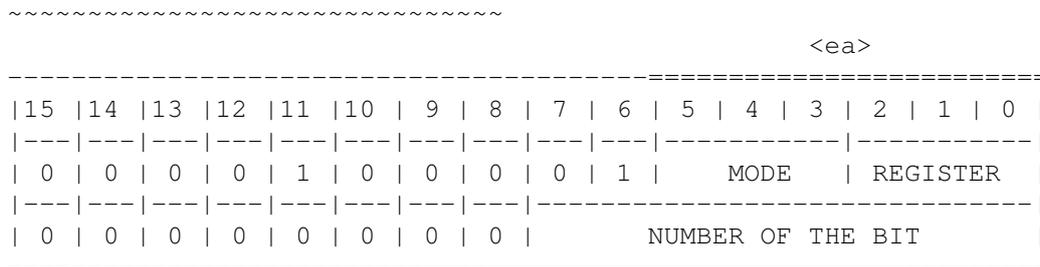
1. Immediate -- The bit number is specified as immediate data.
2. Register -- The specified data register contains the bit number.

FORMAT

In the case of BCHG Dn,<ea>:



In the case of BCHG #<data,<ea>:



REGISTER

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn *	000	N <sup>th</sup> reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N <sup>th</sup> reg. An	(d16,PC)	-	-
(An)+	011	N <sup>th</sup> reg. An	(d8,PC,Xi)	-	-
-(An)	100	N <sup>th</sup> reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N <sup>th</sup> reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N <sup>th</sup> reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N <sup>th</sup> reg. An	#data	-	-

```
|-----|-----|
|([bd,An,Xi]od) |110 |N\textdegree{} reg. An|
|-----|-----|
|([bd,An],Xi,od) |110 |N\textdegree{} reg. An|
|-----|-----|
```

\* Long only; for others modes: Byte only.

RESULT

- X - not affected
- N - not affected
- Z - Set if the bit tested is zero. Cleared otherwise.
- V - not affected
- C - not affected

SEE ALSO

- BCLR BSET BTST
- EOR BFCHG

### 1.41 Bit CLear

NAME

BCLR -- Bit clear

SYNOPSIS

```
BCLR Dn,<ea>
BCLR #<data>,<ea>
```

Size = (Byte, Long)

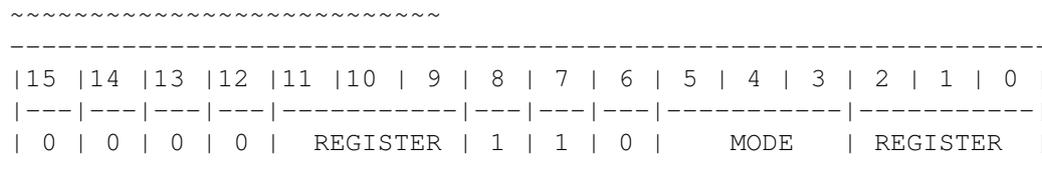
FUNCTION

Tests a bit in the destination operand and sets the Z condition code appropriately, then clears the bit in the destination. If the destination is a data register, any of the 32 bits can be specifice by the modulo 32 number. When the distination is a memory location, the operation must be a byte operation, and therefore the bit number is modulo 8. In all cases, bit zero is the least significant bit. The bit number for this operation may be specified in either of two ways:

1. Immediate -- The bit number is specified as immediate data.
2. Register -- The specified data register contains the bit number.

FORMAT

In the case of BCLR Dn,<ea>:



<ea>

In the case of BCLR #<data>,<ea>:

```
~~~~~
```

<ea>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	1	0	0	0	1	0	MODE		REGISTER				
0	0	0	0	0	0	0	0	NUMBER OF THE BIT								

## REGISTER

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn *	000	$N^{\text{degree}}$ reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	$N^{\text{degree}}$ reg. An	(d16,PC)	-	-
(An)+	011	$N^{\text{degree}}$ reg. An	(d8,PC,Xi)	-	-
-(An)	100	$N^{\text{degree}}$ reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	$N^{\text{degree}}$ reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	$N^{\text{degree}}$ reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	$N^{\text{degree}}$ reg. An	#data	-	-
([bd,An,Xi]od)	110	$N^{\text{degree}}$ reg. An			
([bd,An],Xi,od)	110	$N^{\text{degree}}$ reg. An			

\* Long only; for others modes: Byte only.

## RESULT

X - not affected

N - not affected

Z - Set if the bit tested is zero. Cleared otherwise.

V - not affected

C - not affected

## SEE ALSO

BCHG BSET BTST

AND BFCLR

## 1.42 Bit Field CHanGe

## NAME

BFCHG -- Bit field change

## SYNOPSIS

BFCHG <ea>{OFFSET:WIDTH} (68020+)

No size specs.



## RESULT

-----

X - not affected  
 N - Set if MSB of field is set. Cleared otherwise.  
 Z - Set if all the bits of the field tested are zero.  
 Cleared otherwise.  
 V - Always cleared.  
 C - Always cleared.

## SEE ALSO

BFCLR BFSET

## 1.43 Bit Field CLear

## NAME

BFCLR -- Bit field clear

## SYNOPSIS

BFCLR <ea>{OFFSET:WIDTH} (68020+)

No size specs.

## FUNCTION

<ea> indicates destination operand which a part of bits have to be cleared. Offset enables to locate first bit of field; width specifies number of bits of this field. Field is tested, flags are set, and bits of field are cleared.

Be careful, this instruction operates from MSB to LSB!!

## FORMAT

															<ea>	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
1	1	1	0	1	1	0	0	1	1		MODE		REGISTER			
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
0	0	0	0	Do												
										OFFSET	Dw			WIDTH		

If Do = 0->Field "OFFSET" contains an immediate value which represents effective offset, value from 0 to 31.

If Do = 1->Field "OFFSET" indicates number of a data register (bits 9 and 10 of field cleared) which contains effective offset. Signed value is represented on 32 bits., so it's extended from -2 EXP 31 to (+2 EXP 31) -1.

If Dw = 0->field "WIDTH" contains an immediate value between 1 and 31 which indicates a width from 1 to 31 bits. A value of 0 indicates a width of 32 bits.

If Dw = 1->field "WIDTH" indicates number of a data register (bits 3 and 4 of field cleared) which contains width of bit field. The value modulo 32 can go from 1 to 31, indicating a width from 1 to 31 bits. A value 0 indicates a width of 32 bits.

## REGISTER

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111   001	
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	-	-	(d8,PC,Xi)	-	-
-(An)	-	-	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

## RESULT

X - not affected  
 N - Set if MSB of field is set. Cleared otherwise.  
 Z - Set if all the bits of the field tested are zero.  
 Cleared otherwise.  
 V - Always cleared.  
 C - Always cleared.

## SEE ALSO

BFCHG BFSET

## 1.44 Bit Field Signed EXtract

## NAME

BFEXTS -- Bit field signed extract

## SYNOPSIS

BFEXTS <ea>{OFFSET:WIDTH},Dn (68020+)

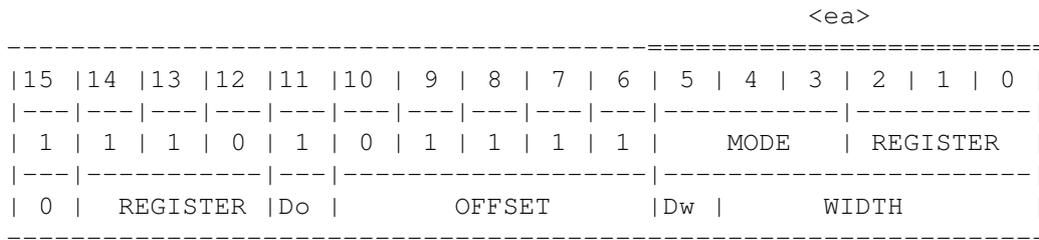
No size specs.

## FUNCTION

<ea> indicates source operand which a part of bits have to be extracted and extended. Offset enables to locate first bit of field; width specifies number of bits of this field. Field is extracted, and extended to 32 bits by the MSB of the field. Result is stored in the data register Dn.

Be careful, this instruction operates from MSB to LSB!!

FORMAT



If Do = 0->Field "OFFSET" contains an immediate value which represents effective offset, value from 0 to 31.

If Do = 1->Field "OFFSET" indicates number of a data register (bits 9 and 10 of field cleared) which contains effective offset. Signed value is represented on 32 bits., so it's extended from -2 EXP 31 to (+2 EXP 31) -1.

If Dw = 0->field "WIDTH" contains an immediate value between 1 and 31 which indicates a width from 1 to 31 bits. A value of 0 indicates a width of 32 bits.

If Dw = 1->field "WIDTH" indicates number of a data register (bits 3 and 4 of field cleared) which contains width of bit field. The value modulo 32 can go from 1 to 31, indicating a width from 1 to 31 bits. A value 0 indicates a width of 32 bits.

REGISTER

<ea> specifies destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	-	-	(d8,PC,Xi)	111	011
-(An)	-	-	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

RESULT

- X - not affected
- N - Set if MSB of field is set. Cleared otherwise.
- Z - Set if all the bits of the field tested are zero. Cleared otherwise.
- V - Always cleared.

C - Always cleared.

SEE ALSO  
BFEXTU

### 1.45 Bit Field Unsigned EXtract

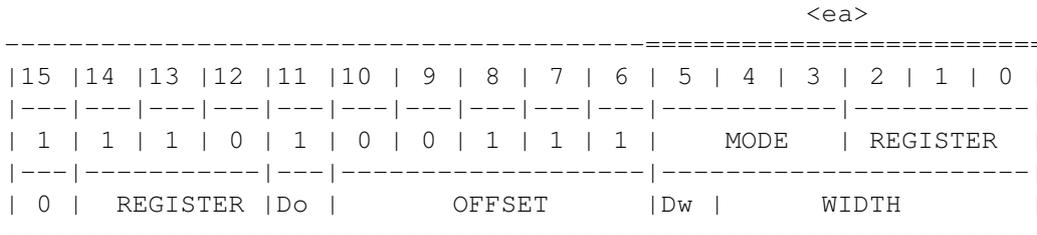
NAME  
BFEXTU -- Bit field unsigned extract"

SYNOPSIS  
BFEXTU <ea>{OFFSET:WIDTH},Dn (68020+)

No size specs.

FUNCTION  
<ea> indicates source operand which a part of bits have to be extracted and extended. Offset enables to locate first bit of field; width specifies number of bits of this field. Field is extracted, and extended to 32 bits by zero. Result is stored in the data register Dn.  
Be careful, this instruction operates from MSB to LSB!!

FORMAT



If Do = 0->Field "OFFSET" contains an immediate value which represents effective offset, value from 0 to 31.

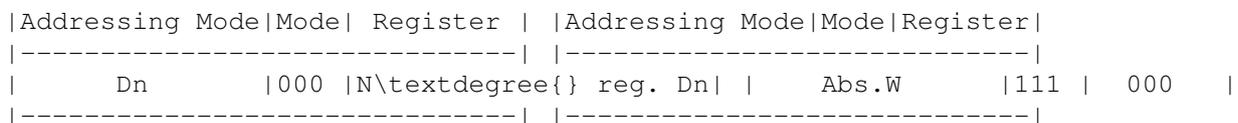
If Do = 1->Field "OFFSET" indicates number of a data register (bits 9 and 10 of field cleared) which contains effective offset. Signed value is represented on 32 bits., so it's extended from -2 EXP 31 to (+2 EXP 31) -1.

If Dw = 0->field "WIDTH" contains an immediate value between 1 and 31 which indicates a width from 1 to 31 bits. A value of 0 indicates a width of 32 bits.

If Dw = 1->field "WIDTH" indicates number of a data register (bits 3 and 4 of field cleared) which contains width of bit field. The value modulo 32 can go from 1 to 31, indicating a width from 1 to 31 bits. A value 0 indicates a width of 32 bits.

REGISTER

<ea> specifies destination, addressing modes are the followings:



An	-	-	Abs.L	111		001	
(An)	010	N\textdegree	reg. An		(d16,PC)	111	010
(An)+	-	-	(d8,PC,Xi)	111		011	
-(An)	-	-	(bd,PC,Xi)	111		011	
(d16,An)	101	N\textdegree	reg. An		([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree	reg. An		([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree	reg. An		#data	-	-
([bd,An,Xi]od)	110	N\textdegree	reg. An				
([bd,An],Xi,od)	110	N\textdegree	reg. An				

## RESULT

X - not affected  
 N - Set if MSB of field is set. Cleared otherwise.  
 Z - Set if all the bits of the field tested are zero.  
 Cleared otherwise.  
 V - Always cleared.  
 C - Always cleared.

## SEE ALSO

BFEXTS

## 1.46 Bit Field Find First One set

## NAME

BFFFO -- Bit field find first one set

## SYNOPSIS

BFFFO <ea>{OFFSET:WIDTH},Dn (68020+)

No size specs.

## FUNCTION

<ea> indicates source operand which a part of bits have to be examined. Offset enables to locate first bit of field; width specifies number of bits of this field. Field is examined, offset of first bit set is encountered, the MSB is stored in the data register Dn.  
 Offset is equal to base offset more offset of bit into examined field.  
 If no bits set are found, data register Dn contains a value which follows to initial offset more width of bit field.  
 Be careful, this instruction operates from MSB to LSB!!

## FORMAT

<ea>

-----

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
-----											-----					
1	1	1	0	1	1	0	1	1	1	MODE		REGISTER				
-----											-----					
0	REGISTER Do			OFFSET					Dw	WIDTH						
-----																

If Do = 0->Field "OFFSET" contains an immediate value which represents effective offset, value from 0 to 31.

If Do = 1->Field "OFFSET" indicates number of a data register (bits 9 and 10 of field cleared) which contains effective offset. Signed value is represented on 32 bits., so it's extended from -2 EXP 31 to (+2 EXP 31) -1.

If Dw = 0->field "WIDTH" contains an immediate value between 1 and 31 which indicates a width from 1 to 31 bits. A value of 0 indicates a width of 32 bits.

If Dw = 1->field "WIDTH" indicates number of a data register (bits 3 and 4 of field cleared) which contains width of bit field. The value modulo 32 can go from 1 to 31, indicating a width from 1 to 31 bits. A value 0 indicates a width of 32 bits.

#### REGISTER

<ea> specifies destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	-	-	(d8,PC,Xi)	111	011
-(An)	-	-	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

#### RESULT

X - not affected

N - Set if MSB of field is set. Cleared otherwise.

Z - Set if all the bits of the field tested are zero.  
Cleared otherwise.

V - Always cleared.

C - Always cleared.

#### SEE ALSO

BFEXTS BFEXTU

### 1.47 Bit Field INsert

NAME

BFINS -- Bit field insert

SYNOPSIS

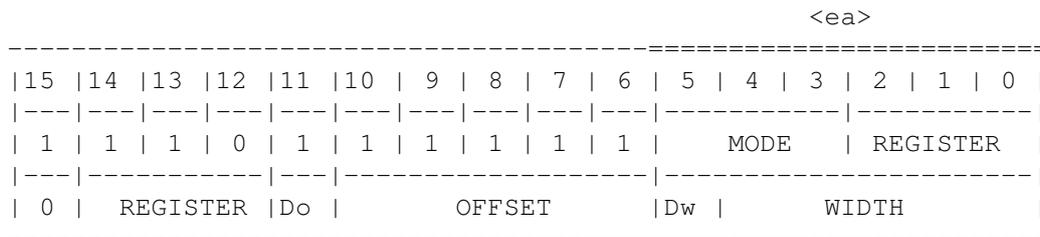
BFINS Dn, <ea>{OFFSET:WIDTH} (68020+)

No size specs.

FUNCTION

<ea> indicates destination operand which field of bits have to be inserted. Offset enables to locate first bit of field; width specifies number of bits of this field. Bit field from Dn, which must be inserted in destination operand, is located in relation to bit zero. This bit zero, after insert, will have for offset: base offset more (width - 1). Be careful, this instruction operates from MSB to LSB!!

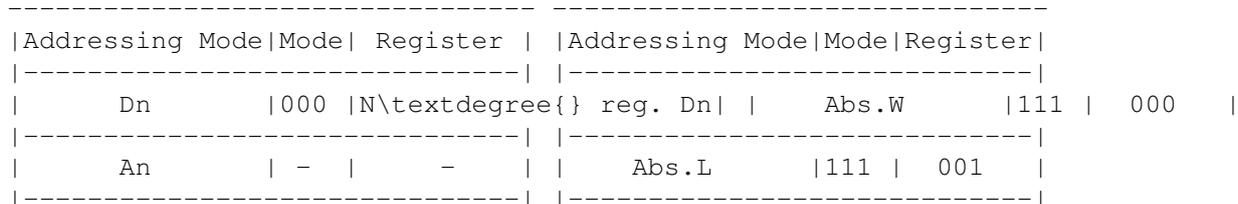
FORMAT



If Do = 0->Field "OFFSET" contains an immediate value which represents effective offset, value from 0 to 31.  
 If Do = 1->Field "OFFSET" indicates number of a data register (bits 9 and 10 of field cleared) which contains effective offset. Signed value is represented on 32 bits., so it's extended from -2 EXP 31 to (+2 EXP 31) -1.  
 If Dw = 0->field "WIDTH" contains an immediate value between 1 and 31 which indicates a width from 1 to 31 bits. A value of 0 indicates a width of 32 bits.  
 If Dw = 1->field "WIDTH" indicates number of a data register (bits 3 and 4 of field cleared) which contains width of bit field. The value modulo 32 can go from 1 to 31, indicating a width from 1 to 31 bits. A value 0 indicates a width of 32 bits.

REGISTER

<ea> specifies destination, addressing modes are the followings:



(An)	010	N\textdegree	reg. An		(d16,PC)		-		-	
(An)+		-		-		(d8,PC,Xi)		-		-
-(An)		-		-		(bd,PC,Xi)		-		-
(d16,An)	101	N\textdegree	reg. An		([bd,PC,Xi],od)		-		-	
(d8,An,Xi)	110	N\textdegree	reg. An		([bd,PC],Xi,od)		-		-	
(bd,An,Xi)	110	N\textdegree	reg. An		#data		-		-	
([bd,An,Xi]od)	110	N\textdegree	reg. An							
([bd,An],Xi,od)	110	N\textdegree	reg. An							

## RESULT

X - not affected  
 N - Set if MSB of field is set. Cleared otherwise.  
 Z - Set if all the bits of the field tested are zero.  
 Cleared otherwise.  
 V - Always cleared.  
 C - Always cleared.

## SEE ALSO

BFEXTS BFEXTU

## 1.48 Bit Field SET

## NAME

BFSET -- Bit field set

## SYNOPSIS

BFSET <ea>{OFFSET:WIDTH} (68020+)

No size specs.

## FUNCTION

<ea> indicates destination operand which a part of bits have to be set. Offset enables to locate first bit of field; width specifies number of bits of this field. Field is tested, flags are set, and bits of field are set.

Be careful, this instruction operates from MSB to LSB!!

## FORMAT

															<ea>				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---				
1	1	1	0	1	1	1	0	1	1	MODE		REGISTER							
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---					
0	0	0	0	Do	OFFSET				Dw	WIDTH									

If Do = 0->Field "OFFSET" contains an immediate value which represents effective offset, value from 0 to 31.

If Do = 1->Field "OFFSET" indicates number of a data register (bits 9 and 10 of field cleared) which contains effective offset. Signed value is represented on 32 bits., so it's extended from -2 EXP 31 to (+2 EXP 31) -1.

If Dw = 0->field "WIDTH" contains an immediate value between 1 and 31 which indicates a width from 1 to 31 bits. A value of 0 indicates a width of 32 bits.

If Dw = 1->field "WIDTH" indicates number of a data register (bits 3 and 4 of field cleared) which contains width of bit field. The value modulo 32 can go from 1 to 31, indicating a width from 1 to 31 bits. A value 0 indicates a width of 32 bits.

#### REGISTER

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	-	-	(d8,PC,Xi)	-	-
-(An)	-	-	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

#### RESULT

X - not affected

N - Set if MSB of field is set. Cleared otherwise.

Z - Set if all the bits of the field tested are zero.  
Cleared otherwise.

V - Always cleared.

C - Always cleared.

#### SEE ALSO

BFCHG BFCLR

## 1.49 Bit Field TeST

NAME

BFTST -- Bit field test

SYNOPSIS

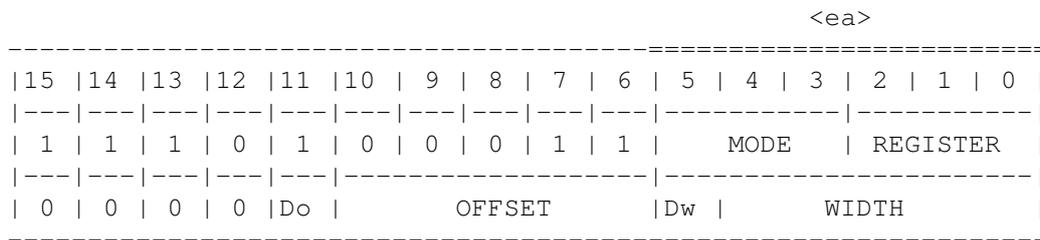
BFTST <ea>{OFFSET:WIDTH} (68020+)

No size specs.

FUNCTION

<ea> indicates destination operand which a part of bits have to be tested. Offset enables to locate first bit of field; width specifies number of bits of this field. Field is tested, flags are set. Be careful, this instruction operates from MSB to LSB!!

FORMAT



If Do = 0->Field "OFFSET" contains an immediate value which represents effective offset, value from 0 to 31.

If Do = 1->Field "OFFSET" indicates number of a data register (bits 9 and 10 of field cleared) which contains effective offset. Signed value is represented on 32 bits., so it's extended from -2 EXP 31 to (+2 EXP 31) -1.

If Dw = 0->field "WIDTH" contains an immediate value between 1 and 31 which indicates a width from 1 to 31 bits. A value of 0 indicates a width of 32 bits.

If Dw = 1->field "WIDTH" indicates number of a data register (bits 3 and 4 of field cleared) which contains width of bit field. The value modulo 32 can go from 1 to 31, indicating a width from 1 to 31 bits. A value 0 indicates a width of 32 bits.

REGISTER

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	-	-	(d8,PC,Xi)	111	011
-(An)	-	-	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011

```

| (d8,An,Xi) |110 |N\textdegree{} reg. An| |([bd,PC],Xi,od)|111 | 011 |
|-----| |-----|
| (bd,An,Xi) |110 |N\textdegree{} reg. An| | #data | - | - |
|-----| |-----|
|([bd,An,Xi]od) |110 |N\textdegree{} reg. An|
|-----|
|([bd,An],Xi,od)|110 |N\textdegree{} reg. An|
|-----|

```

#### RESULT

X - not affected  
 N - Set if MSB of field is set. Cleared otherwise.  
 Z - Set if all the bits of the field tested are zero.  
 Cleared otherwise.  
 V - Always cleared.  
 C - Always cleared.

#### SEE ALSO

BFCHG BFCLR BFSET

## 1.50 Break-Point

#### NAME

BKPT -- Break-point

#### SYNOPSIS

BKPT #<data>

#### FUNCTION

This instruction is used to support the program breakpoint function for debug monitors and real-time hardware emulators, and the operation will be dependent on the implementation. Execution of this instruction will cause the MC68010 to run a breakpoint acknowledge bus cycle and zeros on all address lines, but an MC68020 will place the immediate data on lines A2, A3, and A4, and zeros on lines A0 and A1.

Whether the breakpoint acknowledge cycle is terminated with

DTACK, BERR, or VPA the processor always takes an illegal instruction exception. During exception processing, a debug monitor can distinguish eight different software breakpoints by decoding the field in the BKPT instruction.

For the MC68000 and the MC68HC000, this instruction causes an illegal instruction exception, but does not run the breakpoint acknowledge bus cycle.

There are two possible responses on an MC68020: normal and exception. The normal response is an operation word (typically the instruction

the BKPT originally replaced) on the data lines with the DSACKx signal asserted. The operation word is the executed in place of the breakpoint instruction.

For the exception response, a bus error signal will cause the MC68020 to take an illegal instruction exception, just as an MC68010 or MC68000 would do.

#### FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |N\textdegree{} BREAKPT.|
-----

```

Number of break-point: Value between 0 and 7.

#### RESULT

None.

#### SEE ALSO

ILLEGAL

## 1.51 Unconditional BRAnch

#### NAME

BRA -- Unconditional branch

#### SYNOPSIS

BRA <label>

Size = (Byte, Word)

Size = (Byte, Word, Long) (68020+)

#### FUNCTION

Program execution continues at location (PC) + offset.

Offset is the relative gap between PC value at BRA ((PC) + 2) instruction execution time and mentioned label.

#### FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |           8 BITS OFFSET          |
|-----|
|           16 BITS OFFSET, IF 8 BITS OFFSET = $00           |
|-----|
|           32 BITS OFFSET, IF 8 BITS OFFSET = $FF           |
|-----|

```

#### RESULT

None.

#### SEE ALSO

JMP Bcc

## 1.52 Bit SET

NAME

BSET -- Bit set

SYNOPSIS

BSET Dn,<ea>  
 BSET #<data>,<ea>

Size = (Byte, Long)

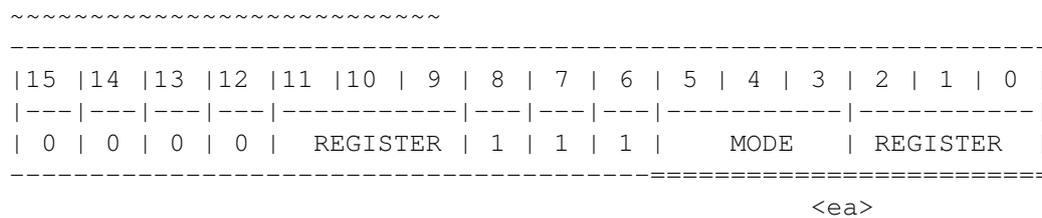
FUNCTION

Tests a bit in the destination operand and sets the Z condition code appropriately, then sets the bit in the destination. If the destination is a data register, any of the 32 bits can be specificed by the modulo 32 number. When the distination is a memory location, the operation must be a byte operation, and therefore the bit number is modulo 8. In all cases, bit zero is the least significant bit. The bit number for this operation may be specified in either of two ways:

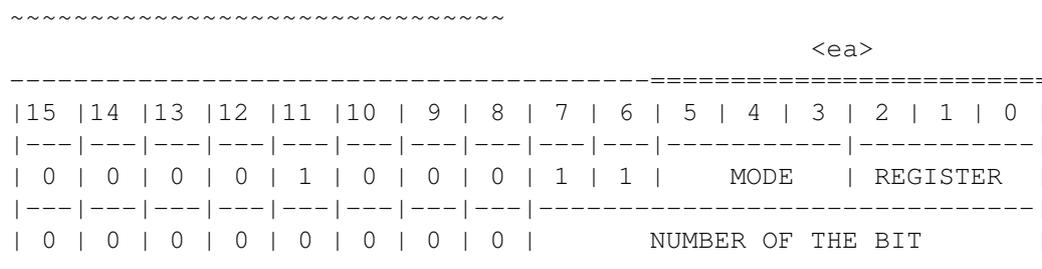
1. Immediate -- The bit number is specified as immediate data.
2. Register -- The specified data register contains the bit number.

FORMAT

In the case of BSET Dn,<ea>:



In the case of BSET #<data,<ea>:



REGISTER

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn *	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-

(An)+	011	N\textdegree{}	reg. An		(d8,PC,Xi)		-		-	
-(An)	100	N\textdegree{}	reg. An		(bd,PC,Xi)		-		-	
(d16,An)	101	N\textdegree{}	reg. An		([bd,PC,Xi],od)		-		-	
(d8,An,Xi)	110	N\textdegree{}	reg. An		([bd,PC],Xi,od)		-		-	
(bd,An,Xi)	110	N\textdegree{}	reg. An		#data		-		-	
([bd,An,Xi]od)	110	N\textdegree{}	reg. An							
([bd,An],Xi,od)	110	N\textdegree{}	reg. An							

\* Long only; for others modes: Byte only.

#### RESULT

X - not affected  
 N - not affected  
 Z - Set if the bit tested is zero. Cleared otherwise.  
 V - not affected  
 C - not affected

#### SEE ALSO

BCHG BCLR BTST  
 OR BFSET BFINS

## 1.53 Branch to SubRoutine

#### NAME

BSR -- Branch to subroutine

#### SYNOPSIS

BSR <label>

Size = (Byte, Word)  
 Size = (Byte, Word, Long) (68020+)

#### FUNCTION

Pushes the long word address which follows the BSR instruction to stack.  
 Program execution continues at location (PC) + offset.  
 Offset is the relative gap between PC value and label.  
 This gap is calculated by complement to two and is coded on 8 bits or on 16 bits.

#### FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	
0	1	1	0	0	0	0	1									

16 BITS OFFSET, IF 8 BITS OFFSET = \$00

32 BITS OFFSET, IF 8 BITS OFFSET = \$FF

-----  
 RESULT  
 None.

SEE ALSO  
 JSR BRA  
 RTS RTD RTR

## 1.54 Bit TeST

NAME  
 BTST -- Bit test

SYNOPSIS  
 BTST Dn,<ea>  
 BTST #<data>,<ea>

Size = (Byte, Long)

### FUNCTION

Tests a bit in the destination operand and sets the Z condition code appropriately. If the destination is a data register, any of the 32 bits can be specified by the modulo 32 number. When the destination is a memory location, the operation must be a byte operation, and therefore the bit number is modulo 8. In all cases, bit zero is the least significant bit. The bit number for this operation may be specified in either of two ways:

1. Immediate -- The bit number is specified as immediate data.
2. Register -- The specified data register contains the bit number.

### FORMAT

In the case of BTST Dn,<ea>:

```

-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0| 0| 0| 0| REGISTER| 1| 0| 0| MODE| REGISTER|
-----
<ea>

```

In the case of BTST #<data>,<ea>:

```

-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0| 0| 0| 0| 1| 0| 0| 0| 0| 0| 0| MODE| REGISTER|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| NUMBER OF THE BIT|
-----

```

REGISTER  
 In the case of BTST Dn,<ea>:

~~~~~

<ea> is always destination, addressing modes are the followings:

| Addressing Mode | Mode | Register               | Addressing Mode | Mode | Register |
|-----------------|------|------------------------|-----------------|------|----------|
| Dn *            | 000  | N\textdegree{} reg. Dn | Abs.W           | 111  | 000      |
| An              | -    | -                      | Abs.L           | 111  | 001      |
| (An)            | 010  | N\textdegree{} reg. An | (d16,PC)        | 111  | 010      |
| (An)+           | 011  | N\textdegree{} reg. An | (d8,PC,Xi)      | 111  | 011      |
| -(An)           | 100  | N\textdegree{} reg. An | (bd,PC,Xi)      | 111  | 011      |
| (d16,An)        | 101  | N\textdegree{} reg. An | ([bd,PC,Xi],od) | 111  | 011      |
| (d8,An,Xi)      | 110  | N\textdegree{} reg. An | ([bd,PC],Xi,od) | 111  | 011      |
| (bd,An,Xi)      | 110  | N\textdegree{} reg. An | #data           | 111  | 100      |
| ([bd,An,Xi]od)  | 110  | N\textdegree{} reg. An |                 |      |          |
| ([bd,An],Xi,od) | 110  | N\textdegree{} reg. An |                 |      |          |

\* Long only; for others modes: Byte only.

In the case of BTST #<data,<ea>:

~~~~~

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn *	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	111	011
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

#### RESULT

X - not affected

N - not affected

Z - Set if the bit tested is zero. Cleared otherwise.

V - not affected  
C - not affected

SEE ALSO

BFTST BFFFO

## 1.55 CALL Module

NAME

CALLM -- Call module (68020 ONLY)

SYNOPSIS

FUNCTION

This instruction is 68020 ONLY and is used with, for cooperation with the PMMU MC68851. Be careful, it's not available on 68030+.

RESULT

SEE ALSO

## 1.56 Compare And Swap

NAME

CAS -- Compare and swap (68020+)

SYNOPSIS

CAS Dc,Du,<ea>

Size = (Byte, Word, Long)

FUNCTION

This instruction is a read-modify-write instruction and should NEVER be used on Amiga because of conflicts with customs chips. Destination operand, which is in memory at address specified by <ea>, is compared to data register Dc (Data Compare). This register is used as reference in the principle of this instruction. If there's equality (Z=1), destination operand can be updated, i.e. the new operand Du (Data Update) is moved in destination. If there's no equality (Z=0), it's the reference register Dc which must be updated. So there's a move from destination operand to Dc.

FORMAT

```

                                                                    <ea>
-----=====
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|-----|---|---|---|-----|-----|
| 0| 0| 0| 0| 1| SIZE  | 0| 1| 1|  MODE  | REGISTER |
|---|---|---|---|---|-----|-----|-----|-----|
| 0| 0| 0| 0| 0| 0| 0|Du REGISTER| 0| 0| 0|Dc REGISTER|
-----=====

```

## SIZE

01->one Byte operation  
 10->one Word operation  
 11->one Long operation

## REGISTER

Du register: indicates number of data register, which contains the new value to update in destination operand.

Dc register: indicates number of data register, which contains the reference value to compare to destination operand.

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111   000	
An	-	-	Abs.L	111   001	
(An)	010	N\textdegree{} reg. An	(d16,PC)	-   -	
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-   -	
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-   -	
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-   -	
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-   -	
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-   -	
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

## RESULT

X - not affected  
 N - Set if the result of comparison is negative. Cleared otherwise.  
 Z - Set if the result of comparison is zero. Cleared otherwise.  
 V - Set if overflow. Cleared otherwise.  
 C - Set if carry. Cleared otherwise.

## SEE ALSO

CAS2

## 1.57 Compare And Swap (two-operand)

## NAME

CAS2 --Compare and swap (two-operand) (68020+)

## SYNOPSIS

CAS2 Dc1:Dc2,Du1:Du2,(Rn1):(Rn2)

Size = (Word, Long)

#### FUNCTION

This instruction is a read-modify-write instruction and should NEVER be used on Amiga because of conflicts with customs chips Destination operand 1, which is in memory to specified address by (Rn1), is compared to data register Dc1. If there's equality (Z=1), destination operand 2, which is in memory to specified address by (Rn2), is compared to data register Dc2.

If there's equality (Z=1), destination operand 1 can be updated, i.e. new operand Du1 (Data Update) is moved in destination (Rn1).

And also, destination operand 2 can be updated, i.e. new operand Du2 (Data Update) is moved in destination (Rn2).

If there's no equality (Z=0), reference registers Dc1 and Dc2 have to be updated. So there's move from destination operand (Rn1) in Dc1, and from (Rn2) in Dc2.

#### FORMAT

```

-----
| 15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|---|---|---|---|-----|---|---|---|---|---|---|---|---|---|
| 0  | 0 | 0 | 0 | 1 | SIZE | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
|----|-----|---|-----|-----|---|---|---|---|-----|
|D/A1|  Rn1 REG. | 0 | 0 | 0 |  Du1 REG. | 0 | 0 | 0 |  Dc1 REG. |
|----|-----|---|---|---|-----|---|---|---|-----|
|D/A2|  Rn2 REG. | 0 | 0 | 0 |  Du2 REG. | 0 | 0 | 0 |  Dc2 REG. |
-----

```

#### SIZE

10->one Word operation

11->one Long operation

#### REGISTER

D/A1 = 0: Rn1=Dn

D/A1 = 1: Rn1=An

D/A2 = 0: Rn2=Dn

D/A2 = 1: Rn2=An

Du1, Du2 registers: indicates number of data register, which contains the new value to update in destination operand.

Dc1, Dc2 registers: indicates number of data register, which contains the reference value to compare to destination operand.

Rn1, Rn2 registers: indicates number of destination registers.

#### RESULT

X - not affected

N - Set following to comparisons results.

Z - Set if the result of comparisons is zero. Cleared otherwise.

V - Set if overflow. Cleared otherwise.

C - Set if carry. Cleared otherwise.

#### SEE ALSO

CAS

## 1.58 CHecK bounds

NAME

CHK -- Check bounds

SYNOPSIS

CHK <ea>,Dn

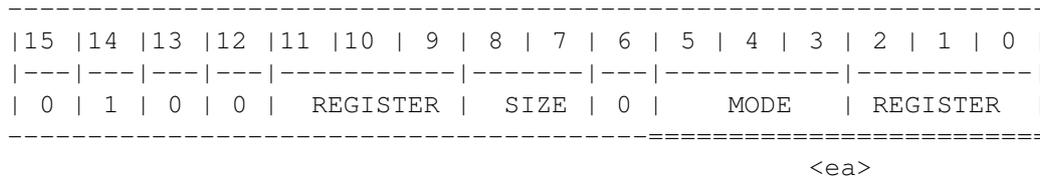
Size = (Word)

Size = (Word, Long) (68020+)

FUNCTION

Compares the value in the data register specified to zero and to the upper bound. The upper bound is a twos complement integer. If the register value is less than zero or greater than the upper bound, a CHK instruction, vector number 6, occurs.

FORMAT



REGISTER

<ea> specifies upper bound, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	111	011
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	111	100
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

SIZE

11->one Word operation  
 10->one Long operation

#### RESULT

X - Not affected  
 N - Set if Dn < 0; cleared if Dn > <ea>. Undefined otherwise.  
 Z - Undefined.  
 V - Undefined.  
 C - Undefined.

#### SEE ALSO

CMP CMPI CMPA  
 CHK2

## 1.59 CHeck register against upper and lower bounds

#### NAME

CHK2 -- Check register against upper and lower bounds (68020+)

#### SYNOPSIS

CHK2 <ea>,Rn

Size = (Byte, Word, Long)

#### FUNCTION

<ea> indicates memory area of two bounds: in 1st memory position, lower bound, in 2nd memory position, upper bound. Those two values are adjacent in memory.

For signed comparisons, the lowest arithmetic value, expressed as a two complement integer, have to be the lower bound.

If Rn is a data register Dn and if size of operation is 8 or 16 bits, only the 8 or 16 bits of low weight of Dn and bounds are taken in count.

In opposite, if Rn is an address register An, it must be extension of bounds sign and 32 bits of An are taken care.

If Rn is located out of 2 bounds, a CHK instruction, vector number 6, occurs.

#### FORMAT

														<ea>		
-----																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
---	---	---	---	---	-----	---	---	---	---	---	---	---	---	---	---	
0	0	0	0	0	SIZE	0	1	1		MODE		REGISTER				
---	-----	---	-----	---	---	---	---	---	---	-----		-----				
D/A	REGISTER	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
-----																

#### REGISTER

Register specifies the register Rn which contains the value to test.

If D/A = 0: Rn = Dn

If D/A = 1: Rn = An

<ea> specifies bounds, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N <sup>textdegree</sup> reg. An	(d16,PC)	111	010
(An)+	-	-	(d8,PC,Xi)	111	011
-(An)	-	-	(bd,PC,Xi)	111	011
(d16,An)	101	N <sup>textdegree</sup> reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N <sup>textdegree</sup> reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N <sup>textdegree</sup> reg. An	#data	-	-
([bd,An,Xi]od)	110	N <sup>textdegree</sup> reg. An			
([bd,An],Xi,od)	110	N <sup>textdegree</sup> reg. An			

#### SIZE

00->one Byte operation  
 01->one Word operation  
 10->one Long operation

#### RESULT

X - Not affected  
 N - Undefined.  
 Z - Set if Rn is equal to one of the two bounds. Cleared otherwise.  
 V - Undefined.  
 C - Set if Rn is out of bounds. Cleared otherwise.

#### SEE ALSO

CMP CMPI CMPA  
 CHK

## 1.60 CLearR

#### NAME

CLR -- Clear

#### SYNOPSIS

CLR <ea>

Size = (Byte, Word, Long)

#### FUNCTION

Clears the destination operand to zero.

On an MC68000 and MC68HC000, a CLR instruction does both a read and a write to the destination. Because of this, this

instruction should never be used on custom chip registers.

#### FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	SIZE			MODE			REGISTER	

<ea>

#### REGISTER

<ea> specifies destination operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N <sup>textdegree</sup> reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N <sup>textdegree</sup> reg. An	(d16,PC)	-	-
(An)+	011	N <sup>textdegree</sup> reg. An	(d8,PC,Xi)	-	-
-(An)	100	N <sup>textdegree</sup> reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N <sup>textdegree</sup> reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N <sup>textdegree</sup> reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N <sup>textdegree</sup> reg. An	#data	-	-
([bd,An,Xi]od)	110	N <sup>textdegree</sup> reg. An			
([bd,An],Xi,od)	110	N <sup>textdegree</sup> reg. An			

#### SIZE

00->one Byte operation  
 01->one Word operation  
 10->one Long operation

#### RESULT

X - Not affected  
 N - Always cleared  
 Z - Always set  
 V - Always cleared  
 C - Always cleared

#### SEE ALSO

MOVE MOVEQ  
 BCLR BFCLR

## 1.61 CoMPare

## NAME

CMP -- Compare

## SYNOPSIS

CMP <ea>,Dn

Size = (Byte, Word, Long)

## FUNCTION

Subtracts the source operand from the destination data register and sets the condition codes according to the result. The data register is NOT changed.

## FORMAT

```

-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|-----|-----|-----|-----|
| 1| 0| 1| 1| REGISTER | OP-MODE | MODE | REGISTER |
=====
<ea>

```

## OP-MODE

000 8 bits operation.  
 001 16 bits operation.  
 010 32 bits operation.

## REGISTER

The data register specifies destination Dn.

<ea> specifies source operand, addressing modes allowed are:

```

-----
|Addressing Mode|Mode| Register | |Addressing Mode|Mode|Register|
|-----|-----|-----| |-----|-----|-----|
|      Dn      |000|N\textdegree{} reg. Dn| | Abs.W      |111| 000 |
|-----|-----|-----| |-----|-----|-----|
|      An *    |001|N\textdegree{} reg. An| | Abs.L      |111| 001 |
|-----|-----|-----| |-----|-----|-----|
|      (An)    |010|N\textdegree{} reg. An| | (d16,PC)   |111| 010 |
|-----|-----|-----| |-----|-----|-----|
|      (An)+   |011|N\textdegree{} reg. An| | (d8,PC,Xi) |111| 011 |
|-----|-----|-----| |-----|-----|-----|
|      -(An)   |100|N\textdegree{} reg. An| | (bd,PC,Xi) |111| 011 |
|-----|-----|-----| |-----|-----|-----|
|      (d16,An)|101|N\textdegree{} reg. An| | ([bd,PC,Xi],od)|111| 011 |
|-----|-----|-----| |-----|-----|-----|
|      (d8,An,Xi)|110|N\textdegree{} reg. An| | ([bd,PC],Xi,od)|111| 011 |
|-----|-----|-----| |-----|-----|-----|
|      (bd,An,Xi)|110|N\textdegree{} reg. An| | #data      |111| 100 |
|-----|-----|-----| |-----|-----|-----|
| ([bd,An,Xi]od)|110|N\textdegree{} reg. An|
|-----|-----|-----|
| ([bd,An],Xi,od)|110|N\textdegree{} reg. An|
-----

```

\* Word and Long only.

## RESULT

X - Not affected

N - Set if the result is negative. Cleared otherwise.  
 Z - Set if the result is zero. Cleared otherwise.  
 V - Set if an overflow occurs. Cleared otherwise.  
 C - Set if a borrow occurs. Cleared otherwise.

SEE ALSO

CMPI CMPA CPM CPM2  
 TST CHK CHK2

## 1.62 CoMPare register against upper and lower bounds

NAME

Cmp2 -- Compare register against upper and lower bounds (68020+)

SYNOPSIS

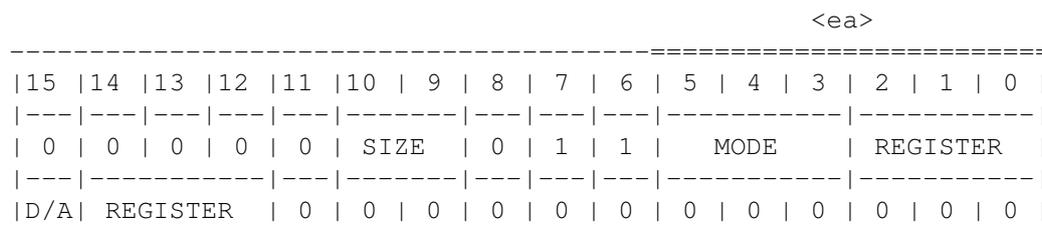
CMP2 <ea>,Rn

FUNCTION

Used to compare value of Rn (Dn or An) with two lower and upper bounds, which are stored in memory, at address given by <ea> (in two adjacent areas).  
 Lower bound have to be stored before upper bound. Flags are set following to the result of comparison.

If Rn is a data register Dn, and if operation is made on 8 or 16 bits, only the 8 or 16 bits of Dn are taken in count.  
 In opposite, in the case of Rn as an address register and if a 16 bits operation is granted, the 32 bits of An are compared to bounds which are, them, extended on 32 bits by their signs.

FORMAT



REGISTER

Register specifies the register Rn which contains the value to test.  
 If D/A = 0: Rn = Dn  
 If D/A = 1: Rn = An  
 <ea> specifies bounds, addressing modes allowed are:

-----														
Addressing Mode	Mode	Register		Addressing Mode	Mode	Register								
-----	-----	-----		-----	-----	-----								
	Dn	-	-		Abs.W	111	000							
	An	-	-		Abs.L	111	001							
	(An)	010	N\textdegree		reg. An		(d16,PC)	111	010					
-----														

(An)+	-	-	(d8,PC,Xi)	111	011
-(An)	-	-	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree	reg. An	111	011
(d8,An,Xi)	110	N\textdegree	reg. An	111	011
(bd,An,Xi)	110	N\textdegree	reg. An	#data	-
([bd,An,Xi]od)	110	N\textdegree	reg. An		
([bd,An],Xi,od)	110	N\textdegree	reg. An		

SIZE

- 00->one Byte operation
- 01->one Word operation
- 10->one Long operation

RESULT

- X - Not affected
- N - Undefined.
- Z - Set if Rn is equal to one of the two bounds. Cleared otherwise.
- V - Undefined.
- C - Set if Rn is out of bounds. Cleared otherwise.

SEE ALSO

- CMP CMPI CMPA
- CHK2

### 1.63 CoMPare Address

NAME

CMPA -- Compare address

SYNOPSIS

CMPA <ea>,An

Size = (Word, Long)

FUNCTION

Subtracts the source operand from the destination address register and sets the condition codes according to the result. The address register is NOT changed. Word sized source operands are sign extended to long for comparison.

FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	REGISTER				OP-MODE			MODE		REGISTER		

<ea>

## OP-MODE

011 16 bits operation.

111 32 bits operation.

## REGISTER

The address register specifies destination An.

<ea> specifies source operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N <sup>th</sup> reg. Dn	Abs.W	111	000
An	001	N <sup>th</sup> reg. An	Abs.L	111	001
(An)	010	N <sup>th</sup> reg. An	(d16,PC)	111	010
(An)+	011	N <sup>th</sup> reg. An	(d8,PC,Xi)	111	011
-(An)	100	N <sup>th</sup> reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N <sup>th</sup> reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N <sup>th</sup> reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N <sup>th</sup> reg. An	#data	111	100
([bd,An,Xi]od)	110	N <sup>th</sup> reg. An			
([bd,An],Xi,od)	110	N <sup>th</sup> reg. An			

## RESULT

X - Not affected

N - Set if the result is negative. Cleared otherwise.

Z - Set if the result is zero. Cleared otherwise.

V - Set if an overflow occurs. Cleared otherwise.

C - Set if a borrow occurs. Cleared otherwise.

## SEE ALSO

CMP CMPI CMP2

## 1.64 CoMPare Immediate

## NAME

CMPI -- Compare immediate

## SYNOPSIS

CMP #<data>, <ea>

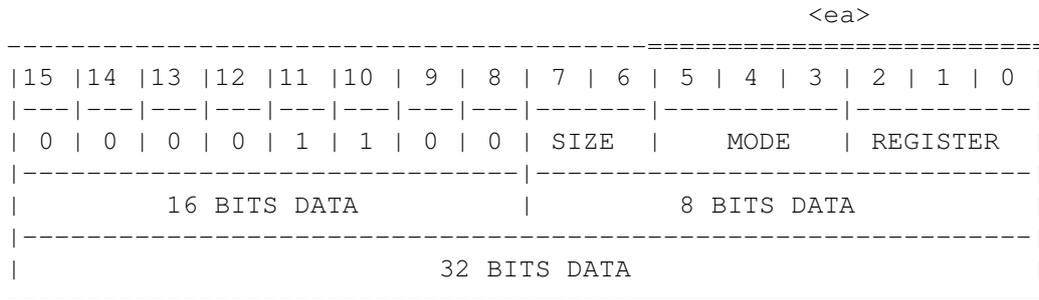
Size = (Byte, Word, Long)

## FUNCTION

Subtracts the source operand from the destination operand and sets the condition codes according to the result. The destination is NOT changed. The size of the immediate data matches the operation

size.

FORMAT



SIZE

- 00->one Byte operation
- 01->one Word operation
- 10->one Long operation

REGISTER

<ea> specifies destination operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	111	011
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

RESULT

- X - Not affected
- N - Set if the result is negative. Cleared otherwise.
- Z - Set if the result is zero. Cleared otherwise.
- V - Set if an overflow occurs. Cleared otherwise.
- C - Set if a borrow occurs. Cleared otherwise.

SEE ALSO

- CMP CMPA CMPM CMP2
- TST CHK CHK2

## 1.65 CoMPare Memory

### NAME

CMPM -- Compare memory

### SYNOPSIS

CMPM (Ay)+, (Ax)+

Size = (Byte, Word, Long)

### FUNCTION

Subtracts the source operand from the destination operand and sets the condition codes according to the result. The destination operand is NOT changed. Operands are always addressed with the postincrement mode.

### FORMAT

```

-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|-----|---|-----|---|---|---|-----|
| 1| 0| 1| 1|Ax REGISTER| 1| SIZE | 0| 0| 1|Ay REGISTER|
-----=====
                                     <ea>

```

### SIZE

00->one Byte operation

01->one Word operation

10->one Long operation

### REGISTER

Ax register specifies destination operand (for post-incrementation).

Ay register specifies source operand.

### RESULT

X - Not affected

N - Set if the result is negative. Cleared otherwise.

Z - Set if the result is zero. Cleared otherwise.

V - Set if an overflow occurs. Cleared otherwise.

C - Set if a borrow occurs. Cleared otherwise.

### SEE ALSO

CMP CMPI CMPA CMP2

TST CHK CHK2

## 1.66 Branch on CoProcessor condition

### NAME

cpBcc -- Branch on coprocessor condition

### SYNOPSIS

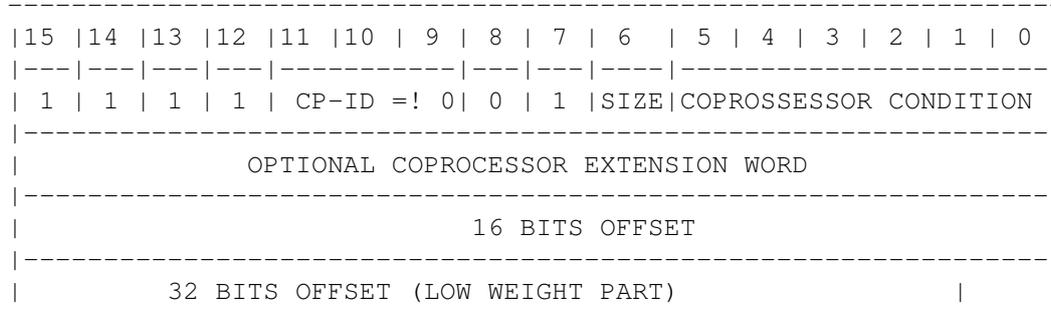
cpBcc <label>

Offset size = (Word, Long)

## FUNCTION

If specified coprocessor condition is true, program execution continues to address pointed by searching PC more offset. Searching PC contains address of first word of offset. Offset is a signed value of 16 or 32 bits, which represents the relative gap in bytes between the searching PC and destination address.

## FORMAT



## SIZE

0->one Word operation  
 1->one Long operation

CP-ID field identify coprocessor (1 to 7). If CP-ID=0, "line emulation F" exception is generated.

"COPROSSESSOR CONDITION" field, specifies condition to test. This condition is addressed to coprocessor which, after examining this one, address directives to processor in order to execute the instruction.

## RESULT

Not affected.

## SEE ALSO

cpDBcc

## 1.67 Decrement and Branch on CoProcessor condition

## NAME

cpDBcc -- Decrement and branch on coprocessor condition

## SYNOPSIS

cpDBcc Dn,<label>

Offset size = (Word)

## FUNCTION

If specified coprocessor condition is true, program execution continues with next instruction. Else 16 bits of data register which are used as a counter, are decremented of one. If Dn = -1, execution continues with next instruction. If Dn != -1, execution continues to address pointed by searching PC more offset; searching PC containing address of first word of offset.

Offset is a signed value of 16 bits, which represents the relative gap in bytes between the searching PC and destination address.

#### FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	1	1	CP-ID =!				0	0	0	1	0	0	1	REGISTER	
0	0	0	0	0	0	0	0	0	0	COPROCESSOR CONDITION						
OPTIONAL COPROCESSOR EXTENSION WORD																
16 BITS OFFSET																

CP-ID field identify coprocessor (1 to 7). If CP-ID=0, "line emulation F" exception is generated.

"COPROCESSOR CONDITION" field, specifies condition to test. This condition is addressed to coprocessor which, after examining this one, address directives to processor in order to execute the instruction.

Register field indicates the number of data register used as counter.

#### RESULT

Not affected.

#### SEE ALSO

cpBcc

## 1.68 GENERAL CoProcessor instruction

#### NAME

cpGEN -- General coprocessor instruction

#### SYNOPSIS

cpGEN <coprocessor defined parameters>

No size specs.

#### FUNCTION

Command word specified is addressed to selected coprocessor. This one decode the command and address requests services to processor in order to execute services asked by the coprocessor.

#### FORMAT

															<ea>
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID =!				0	0	0	0	MODE		REGISTER	
COPROCESSOR COMMAND WORD															

```

|-----|
|   OPTIONAL COPROCESSOR EXTENSION WORD or OPTIONAL <ea>   |
|-----|
    
```

CP-ID field identify coprocessor (1 to 7). If CP-ID=0, "line emulation F" exception is generated.

<ea> field specifies place of operand not resident in the coprocessor. Allowed addressing modes are driven by the type of operation made.

"COPROCESSOR COMMAND" field specifies operation to do. This word is transmit to coprocessor which, after analyse of this word, send directives to coprocessor in order to execute instruction.

RESULT

Not affected or modified by the coprocessor.

SEE ALSO

cpBcc

### 1.69 RESTORE CoProcessor instruction (PRIVILEGED)

NAME

cpRESTORE -- Restore coprocessor instruction (PRIVILEGED)

SYNOPSIS

cpRESTORE <ea>

No size specs.

FUNCTION

Restore internal state of coprocessor after it has been generally saved by cpSAVE.

FORMAT

```

                                                                    <ea>
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|-----|---|---|---|-----|-----|
| 1| 1| 1| 1| CP-ID =! 0| 1| 0| 1|   MODE   | REGISTER |
-----
    
```

CP-ID field identify coprocessor (1 to 7). If CP-ID=0, "line emulation F" exception is generated.

REGISTER

<ea> field specifies place where is stored coprocessor context in memory.

Allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001

(An)	010	N\textdegree{}	reg. An		(d16,PC)	111		010	
(An)+	011	N\textdegree{}	reg. An		(d8,PC,Xi)	111		011	
-(An)	-	-			(bd,PC,Xi)	111		011	
(d16,An)	101	N\textdegree{}	reg. An		([bd,PC,Xi],od)	111		011	
(d8,An,Xi)	110	N\textdegree{}	reg. An		([bd,PC],Xi,od)	111		011	
(bd,An,Xi)	110	N\textdegree{}	reg. An		#data	-		-	
([bd,An,Xi]od)	110	N\textdegree{}	reg. An						
([bd,An],Xi,od)	110	N\textdegree{}	reg. An						

RESULT  
Not affected.

SEE ALSO  
cpSAVE

### 1.70 SAVE CoProcessor instruction (PRIVILEGED)

NAME  
cpSAVE -- Save coprocessor instruction (PRIVILEGED)

SYNOPSIS  
cpSAVE <ea>

No size specs.

FUNCTION  
Save internal state of coprocessor.

FORMAT  
<ea>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	
1	1	1	1	CP-ID	!=	0	1	0	0		MODE		REGISTER			

CP-ID field identify coprocessor (1 to 7). If CP-ID=0, "line emulation F" exception is generated.

REGISTER  
<ea> field specifies place where is stored coprocessor context in memory.  
Allowed addressing modes are:

Addressing Mode Mode  Register	Addressing Mode Mode Register
-----	-----

Dn	-	-	Abs.W	111		000	
An	-	-	Abs.L	111		001	
(An)	010	N\textdegree	reg. An		(d16,PC)		-   -
(An)+	011	N\textdegree	reg. An		(d8,PC,Xi)		-   -
-(An)	-	-	(bd,PC,Xi)		-		-
(d16,An)	101	N\textdegree	reg. An		([bd,PC,Xi],od)		-   -
(d8,An,Xi)	110	N\textdegree	reg. An		([bd,PC],Xi,od)		-   -
(bd,An,Xi)	110	N\textdegree	reg. An		#data		-   -
([bd,An,Xi]od)	110	N\textdegree	reg. An				
([bd,An],Xi,od)	110	N\textdegree	reg. An				

RESULT  
Not affected.

SEE ALSO  
cpRESTORE

### 1.71 Set one byte on CoProcessor condition

NAME  
cpScc -- Set one byte on coprocessor condition

SYNOPSIS  
cpScc <ea>

Size = (Byte)

FUNCTION  
If given condition is true, byte specified by <ea> is loaded with \$FF.  
Else it is loaded with \$00.

FORMAT

															<ea>
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1												
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	OPTIONAL COPROCESSOR EXTENSION WORD or OPTIONAL <ea>														
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

CP-ID field identify coprocessor (1 to 7). If CP-ID=0, "line emulation F" exception is generated.

"COPROCESSOR CONDITION" field, specifies condition to test. This condition is addressed to coprocessor which, after examining this one, address directives to processor in order to execute the instruction.

#### REGISTER

<ea> field specifies place of destination byte.  
Allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

#### RESULT

Not affected.

#### SEE ALSO

cpBcc

## 1.72 Exception generation on CoProcessor condition

#### NAME

cpTRAPcc -- Exception generation on coprocessor condition

#### SYNOPSIS

```
cpTRAPcc
cpTRAPcc #<data>
```

No size specs, or size of data: (Word, Long).

#### FUNCTION

If specified coprocessor condition is true, exception vector n\textdegree{}7 is generated. Value of saved PC is next instruction one (return address).

If given condition is false, PC takes the value of next instruction.



address.

Condition code 'cc' specifies one of the following:

0000	F	False	Z = 1	1000	VC	oVerflow Clear	V = 0
0001	T	True	Z = 0	1001	VS	oVerflow Set	V = 1
0010	HI	HIgh	C + Z = 0	1010	PL	PLus	N = 0
0011	LS	Low or Same	C + Z = 1	1011	MI	MInus	N = 1
0100	CC	Carry Clear	C = 0	1100	GE	Greater or Equal	N (+) V = 0
0101	CS	Carry Set	C = 1	1101	LT	Less Than	N (+) V = 1
0110	NE	Not Equal	Z = 0	1110	GT	Greater Than	Z + (N (+) V) = 0
0111	EQ	EQual	Z = 1	1111	LE	Less or Equal	Z + (N (+) V) = 1

Keep the following in mind when using DBcc instructions:

1. A DBcc acts as the UNTIL loop construct in high level languages. E.g., DBMI would be "decrement and branch until minus".
2. Most assemblers accept DBRA or DBF for use when no condition is required for termination of a loop.

The DBcc will, unlike the Bcc instruction, take the branch only if the set of conditions are NOT satisfied. The loop will be terminated if the condition is true, or the counter is zero BEFORE a decrement, and wrap to -1. This mean that if you execute code like:

```
move.w #30,d0
```

```
.Loop
move.l (a0)+,(a1)+
dbf d0,.Loop
```

then the copy will be executed \*31\* times, and 124 bytes of memory will be copied, not 120.

A good practice is therefore to write:

```
move.w #31-1,d0
```

```
.Loop
move.l (a0)+,(a1)+
dbf d0,.Loop
```

To compare two strings that may be in excess of 64k length for beeing equal, you could execute the following code:

...

```
move.l #$53452-1,d0
beq.s .NoLength ; Zero length!
bra.s .StartOuterLoop ; The upper word contain count of 65536's...
```

```
.OuterLoop
swap d0
```

```
.InnerLoop
cmp.b (a0)+,(a1)+
dbne d0,.InnerLoop ; Remember, drop trough on condition TRUE.
```

```
.StartOuterLoop          ; d0 will be $FFFF on 2.+ run-through
  bne.s .NotEqual ; Dropped through due to Not Equal?
  swap d0          ; Get upper part of word...
  dbf d0,.OuterLoop
  ...
```

It would not be possible to use two sets of DBNEs, as SWAP changes the condition codes - and we don't want the drop-though to be on account of D0, instead of the compares...

Another neat trick is to use instruction as a conditional decrementer; this code will decrement d0.w if the last instruction returned NOT EQUAL:

```
...
dbeq d0,.Next
```

```
.Next
...
```

#### FORMAT

```
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|-----|---|---|---|---|---|-----|
| 0| 1| 0| 1|  CONDITION  | 1| 1| 0| 0| 1| REGISTER  |
|-----|
|                                     16 BITS OFFSET (d16)                                     |
|-----
```

"CONDITION" is one of the condition code given some lines before.  
"REGISTER" is the number of data register.  
Offset is the relative gap in byte to do branching.

#### RESULT

Not affected.

#### SEE ALSO

Bcc Scc

## 1.74 Signed DIVide

#### NAME

DIVS, DIVSL -- Signed divide

#### SYNOPSIS

```
DIVS.W <ea>,Dn      32/16 -> 16r:16q
DIVS.L <ea>,Dq      32/32 -> 32q      (68020+)
DIVS.L <ea>,Dr:Dq   64/32 -> 32r:32q  (68020+)
DIVSL.L <ea>,Dr:Dq  32/32 -> 32r:32q  (68020+)
```

Size = (Word, Long)

#### FUNCTION

Divides the signed destination operand by the signed source operand and stores the signed result in the destination.

The instruction has a word form and three long forms. For the word form, the destination operand is a long word and the source operand is a word. The resultant quotient is placed in the lower word of the destination and the resultant remainder is placed in the upper word of the destination. The sign of the remainder is the same as the sign of the dividend.

In the first long form, the destination and the source are both long words. The quotient is placed in the longword of the destination and the remainder is discarded.

The second long form has the destination as a quadword (eight bytes), specified by any two data registers, and the source is a long word. The resultant remainder and quotient are both long words and are placed in the destination registers.

The final long form has both the source and the destination as long words and the resultant quotient and remainder as long words.

#### FORMAT

In the case of DIVS.W:

```

~~~~~<ea>
=====
|15|14|13|12|11|10|9|8|7|6|5|4|3|2|1|0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|1|0|0|0|REGISTER|1|1|1|MODE|REGISTER|
-----

```

"REGISTER" indicates the number of data register.

In the case of DIVS.L and of DIVL.L:

```

~~~~~<ea>
=====
|15|14|13|12|11|10|9|8|7|6|5|4|3|2|1|0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|0|1|0|0|1|1|0|0|0|1|MODE|REGISTER|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|0|Dq REGISTER|1|SIZE|0|0|0|0|0|0|0|0|Dr REGISTER|
-----

```

"Dq REGISTER" indicates the number of data register for destination operand. This register first contains 32 bits of low weight of dividend, and after the value of quotient on 32 bits.

"SIZE" specifies if dividend is on 32 or 64 bits:

0-> 32 bits dividend placed in Dq.

1-> 64 bits dividend placed in Dr:Dq.

"Dr REGISTER" indicates the number of data register for destination operand. This register first contains 32 bits of upper weight of dividend if "SIZE" = 1, and after the value of rest on 32 bits.

If Dr and Dq represents the same register, only quotient on 32 bits is put in Dq.

<ea> field specifies source operand, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	$\text{N}\text{degree}$ reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	$\text{N}\text{degree}$ reg. An	(d16,PC)	111	010
(An)+	011	$\text{N}\text{degree}$ reg. An	(d8,PC,Xi)	111	011
-(An)	100	$\text{N}\text{degree}$ reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	$\text{N}\text{degree}$ reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	$\text{N}\text{degree}$ reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	$\text{N}\text{degree}$ reg. An	#data	111	100
([bd,An,Xi]od)	110	$\text{N}\text{degree}$ reg. An			
([bd,An],Xi,od)	110	$\text{N}\text{degree}$ reg. An			

#### RESULT

- X - Not affected
- N - Set if the quotient is negative, cleared otherwise. Undefined if overflow or divide by zero occurs.
- Z - Set if the quotient is zero, cleared otherwise. Undefined if overflow or divide by zero occurs.
- V - Set if overflow occurs, cleared otherwise. Undefined if divide by zero occurs.
- C - Always cleared.

#### Notes:

1. If divide by zero occurs, an exception occurs.
2. If overflow occurs, neither operand is affected.

#### SEE ALSO

DIVU MULS MULU

## 1.75 Unsigned DIVide

#### NAME

DIVU, DIVUL -- Unsigned divide

#### SYNOPSIS

```

DIVU.W <ea>,Dn      32/16 -> 16r:16q
DIVU.L <ea>,Dq      32/32 -> 32q      (68020+)
DIVU.L <ea>,Dr:Dq   64/32 -> 32r:32q   (68020+)
DIVUL.L <ea>,Dr:Dq  32/32 -> 32r:32q   (68020+)

```

Size = (Word, Long)

## FUNCTION

Divides the unsigned destination operand by the unsigned source operand and stores the unsigned result in the destination.

The instruction has a word form and three long forms. For the word form, the destination operand is a long word and the source operand is a word. The resultant quotient is placed in the lower word of the destination and the resultant remainder is placed in the upper word of the destination. The sign of the remainder is the same as the sign of the dividend.

In the first long form, the destination and the source are both long words. The quotient is placed in the longword of the destination and the remainder is discarded.

The second long form has the destination as a quadword (eight bytes), specified by any two data registers, and the source is a long word. The resultant remainder and quotient are both long words and are placed in the destination registers.

The final long form has both the source and the destination as long words and the resultant quotient and remainder as long words.

## FORMAT

In the case of DIVU.W:

```

~~~~~<ea>
=====
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1| 0| 0| 0| REGISTER | 0| 1| 1|   MODE   | REGISTER |
=====

```

"REGISTER" indicates the number of data register.

In the case of DIVU.L and of DIVUL.L:

```

~~~~~<ea>
=====
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0| 1| 0| 0| 1| 1| 0| 0| 0| 1|   MODE   | REGISTER |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0|Dq REGISTER| 0|SIZE| 0| 0| 0| 0| 0| 0| 0| 0| 0|Dr REGISTER|
=====

```

"Dq REGISTER" indicates the number of data register for destination operand. This register first contains 32 bits of low weight of dividend, and after the value of quotient on 32 bits.

"SIZE" specifies if dividend is on 32 or 64 bits:

0-> 32 bits dividend placed in Dq.

1-> 64 bits dividend placed in Dr:Dq.

"Dr REGISTER" indicates the number of data register for destination operand. This register first contains 32 bits of upper weight of dividend if "SIZE" = 1, and after the value of rest on 32 bits.

If Dr and Dq represents the same register, only quotient on 32 bits

is put in Dq.

<ea> field specifies source operand, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	111	011
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	111	100
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

## RESULT

X - Not affected

N - See below.

Z - Set if the quotient is zero, cleared otherwise. Undefined if overflow or divide by zero occurs.

V - Set if overflow occurs, cleared otherwise. Undefined if divide by zero occurs.

C - Always cleared.

### Notes:

1. If divide by zero occurs, an exception occurs.

2. If overflow occurs, neither operand is affected.

According to the Motorola data books, the N flag is set if the quotient is negative, but in an unsigned divide, this seems to be impossible.

## SEE ALSO

DIVS MULS MULU

## 1.76 Exclusive logical OR

### NAME

EOR -- Exclusive logical OR

### SYNOPSIS

EOR Dn, <ea>

Size = (Byte, Word, Long)

#### FUNCTION

Performs an exclusive OR operation on the destination operand with the source operand.

#### FORMAT

```

-----
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ---| ---| ---| ---| -----| -----| -----| -----|
| 1  | 0  | 1  | 1  | REGISTER | OP-MODE | MODE | REGISTER |
-----=====
                                     <ea>

```

#### OP-MODE

100 8 bits operation.  
 101 16 bits operation.  
 110 32 bits operation.

#### REGISTER

The data register specifies source Dn.  
 <ea> specifies destination operand, addressing modes allowed are:

```

-----
|Addressing Mode|Mode| Register | |Addressing Mode|Mode|Register|
|-----|-----|-----| |-----|-----|-----|
|      Dn      |000| N\textdegree{} reg. Dn| |  Abs.W      |111| 000 |
|-----|-----|-----| |-----|-----|-----|
|      An      | - | - | |  Abs.L      |111| 001 |
|-----|-----|-----| |-----|-----|-----|
|      (An)     |010| N\textdegree{} reg. An| | (d16,PC)   | - | - |
|-----|-----|-----| |-----|-----|-----|
|      (An)+    |011| N\textdegree{} reg. An| | (d8,PC,Xi) | - | - |
|-----|-----|-----| |-----|-----|-----|
|      -(An)    |100| N\textdegree{} reg. An| | (bd,PC,Xi) | - | - |
|-----|-----|-----| |-----|-----|-----|
|      (d16,An) |101| N\textdegree{} reg. An| | ([bd,PC,Xi],od) | - | - |
|-----|-----|-----| |-----|-----|-----|
|      (d8,An,Xi) |110| N\textdegree{} reg. An| | ([bd,PC],Xi,od) | - | - |
|-----|-----|-----| |-----|-----|-----|
|      (bd,An,Xi) |110| N\textdegree{} reg. An| | #data      | - | - |
|-----|-----|-----| |-----|-----|-----|
| ([bd,An,Xi]od) |110| N\textdegree{} reg. An|
|-----|-----|-----|
| ([bd,An],Xi,od) |110| N\textdegree{} reg. An|
-----

```

#### RESULT

X - Not Affected  
 N - Set to the value of the most significant bit.  
 Z - Set if the result is zero.  
 V - Always cleared  
 C - Always cleared

#### SEE ALSO

EORI BCHG

## 1.77 Exclusive OR Immediate

### NAME

EORI -- Exclusive OR immediate

### SYNOPSIS

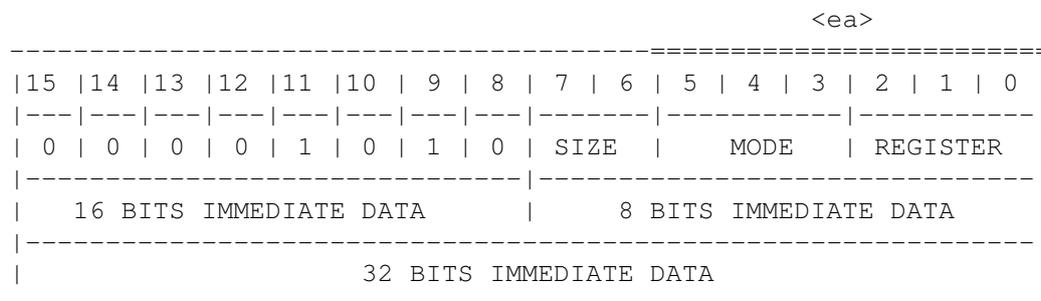
EORI #&lt;data&gt;, &lt;ea&gt;

Size = (Byte, Word, Long)

### FUNCTION

Performs an exclusive OR operation on the destination operand with the source operand.

### FORMAT



### SIZE

00-&gt;8 bits operation.

01-&gt;16 bits operation.

10-&gt;32 bits operation.

### REGISTER

Immediate data is placed behind the word of operating code of the instruction on 8, 16 or 32 bits.

<ea> specifies destination operand, addressing modes allowed are:

Addressing Mode		Mode	Register	Addressing Mode		Mode	Register
-----							
Dn		000	N\textdegree{} reg. Dn		Abs.W	111	000
-----							
An	-	-		Abs.L	111	001	
-----							
(An)		010	N\textdegree{} reg. An		(d16,PC)	-	-
-----							
(An)+		011	N\textdegree{} reg. An		(d8,PC,Xi)	-	-
-----							
-(An)		100	N\textdegree{} reg. An		(bd,PC,Xi)	-	-
-----							
(d16,An)		101	N\textdegree{} reg. An		([bd,PC,Xi],od)	-	-
-----							
(d8,An,Xi)		110	N\textdegree{} reg. An		([bd,PC],Xi,od)	-	-
-----							

```

| (bd,An,Xi) |110 |N\textdegree{} reg. An| | #data | - | - |
|-----|-----|
| ([bd,An,Xi]od) |110 |N\textdegree{} reg. An|
|-----|
| ([bd,An],Xi,od) |110 |N\textdegree{} reg. An|
|-----|

```

## RESULT

X - Not Affected  
N - Set to the value of the most significant bit.  
Z - Set if the result is zero.  
V - Always cleared  
C - Always cleared

## SEE ALSO

EOR EORI to CCR EORI to SR  
BCHG

## 1.78 Exclusive OR Immediate to CCR

## NAME

EORI to CCR -- Exclusive OR immediate to the condition code register

## SYNOPSIS

EORI #<data>,CCR

Size = (Byte)

## FUNCTION

Performs an exclusive OR operation on the condition codes register with the source operand.

## FORMAT

```

-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 BITS IMMEDIATE DATA |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

```

## RESULT

X - Changed if bit 4 of the source is set, cleared otherwise.  
N - Changed if bit 3 of the source is set, cleared otherwise.  
Z - Changed if bit 2 of the source is set, cleared otherwise.  
V - Changed if bit 1 of the source is set, cleared otherwise.  
C - Changed if bit 0 of the source is set, cleared otherwise.

## SEE ALSO

EOR EORI EORI to SR

## 1.79 Exclusive OR immediated with SR (PRIVILEGED)

## NAME

EORI to SR -- Exclusive OR immediated to the status register (PRIVILEGED)

## SYNOPSIS

EORI #<data>,SR

Size = (Word)

## FUNCTION

Performs an exclusive OR operation on the status register with the source operand.

## FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
|-----|
|                                     16 BITS IMMEDIATE DATA                                     |
-----

```

## RESULT

X - Changed if bit 4 of the source is set, cleared otherwise.  
 N - Changed if bit 3 of the source is set, cleared otherwise.  
 Z - Changed if bit 2 of the source is set, cleared otherwise.  
 V - Changed if bit 1 of the source is set, cleared otherwise.  
 C - Changed if bit 0 of the source is set, cleared otherwise.

## SEE ALSO

EOR EORI EORI to CCR

## 1.80 Register EXchanGe

## NAME

EXG -- Register exchange

## SYNOPSIS

EXG Rx,Ry

Size = (Long)

## FUNCTION

Exchanges the contents of any two registers.

## FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|-----|---|-----|-----|
| 1 | 1 | 0 | 0 |Rx REGISTER| 1 |           OP-MODE           |Ry REGISTER|
-----

```

"Rx REGISTER" specifies a data or address register. If it's an exchange between a data register and an address register, this field define the data register.

"Ry REGISTER" specifies a data or address register. If it's an exchange between a data register and an address register, this field define the address register.

#### OP-MODE

01000->Exchange between data registers.  
 01001->Exchange between address registers.  
 10001->Exchange between data and address registers.

#### RESULT

None.

#### SEE ALSO

SWAP

## 1.81 Sign EXTend

#### NAME

EXT, EXTB -- Sign extend

#### SYNOPSIS

EXT.W Dn Extend byte to word  
 EXT.L Dn Extend word to long word  
 EXTB.L Dn Extend byte to long word (68020+)

Size = (Word, Long)

#### FUNCTION

Extends a byte to a word, or a word to a long word in a data register by copying the sign bit through the upper bits. If the operation is from byte to word, bit 7 is copied to bits 8 through 15. If the operation is from word to long word, bit 15 is copied to bits 16 through 31. The EXTB copies bit 7 to bits 8 through 31.

#### FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|-----|---|---|---|-----|
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | OP-MODE | 0 | 0 | 0 | REGISTER |
-----
```

"REGISTER" specifies a data register.

#### OP-MODE

010->Extending from 8 bits to 16 bits.  
 011->Extending from 16 bits to 32 bits.  
 111->Extending from 8 bits to 32 bits.

#### RESULT

X - Not affected  
 N - Set if the result is negative. Cleared otherwise.  
 Z - Set if the result is zero. Cleared otherwise.  
 V - Always cleared  
 C - Always cleared

SEE ALSO  
BFEXTS BFEXTU

## 1.82 Illegal processor instruction

NAME  
ILLEGAL -- Illegal processor instruction

SYNOPSIS  
ILLEGAL

FUNCTION  
This instruction forces an Illegal Instruction exception, vector number 4. All other illegal instruction bit patterns, including, but not limited to, \$fxxx and \$axxx, are reserved for future expansion.  
The Hexa-decimal code of this instruction is: \$4AFC

FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
-----
```

RESULT  
None.

SEE ALSO  
BKPT

## 1.83 Unconditional far JuMP

NAME  
JMP -- Unconditional far jump

SYNOPSIS  
JMP <ea>

FUNCTION  
Program execution continues at the address specified by the operand.

FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|-----|-----|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |   MODE   | REGISTER |
-----=====
                                     <ea>
```

REGISTER

<ea> specifies address of next instruction.  
 Allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	-	-	(d8,PC,Xi)	111	011
-(An)	-	-	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

RESULT  
 None.

SEE ALSO  
 BRA JSR

### 1.84 Jump to far SubRoutine

NAME  
 JSR -- Jump to far subroutine

SYNOPSIS  
 JSR <ea>

FUNCTION  
 Pushes the long word address of the instruction immediately following the JSR instruction onto the stack. The PC contains the address of the instruction word plus two. Program execution continues at location specified by <ea>.

FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
0	1	0	0	1	1	1	0	1	0		MODE			REGISTER	

<ea>

REGISTER

<ea> specifies address of next instruction.  
Allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	-	-	(d8,PC,Xi)	111	011
-(An)	-	-	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

#### RESULT

None.

#### SEE ALSO

BSR BRA  
RTS RTD RTR

## 1.85 Load Effective Address

#### NAME

LEA -- Load effective address

#### SYNOPSIS

LEA <ea>,An

Size = (Long)

#### FUNCTION

Places the specified address into the destination address register. Note: All 32 bits of An are affected by this instruction.

#### FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
---	---	---	---	-----	---	---	---	-----	---	---	---	---	---	---	---
0	1	0	0	REGISTER	1	1	1	MODE	REGISTER						

<ea>

## REGISTER

"REGISTER" indicates the number of address register

<ea> specifies address which must be loaded in the address register.  
Allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	-	-	(d8,PC,Xi)	111	011
-(An)	-	-	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

## RESULT

None.

## SEE ALSO

MOVEA ADDA SUBA

## 1.86 Create local stack frame

## NAME

LINK -- Create local stack frame

## SYNOPSIS

LINK An, #<data>

Size = (Word)

Size = (Word, Long) (68020+)

## FUNCTION

This instruction saves the specified address register onto the stack, then places the new stack pointer in that register. It then adds the specified immediate data to the stack pointer. To allocate space on the stack for a local data area, a negative value should be used for the second operand.

The use of a local stack frame is critically important to the

programmer who wishes to write re-entrant or recursive functions. The creation of a local stack frame on the MC680x0 family is done through the use of the LINK and UNLK instructions. The LINK instruction saves the frame pointer onto the stack, and places a pointer to the new stack frame in it. The UNLK instruction restores the old stack frame. For example:

```
link  a5,#-8    ; a5 is chosen to be the frame
        ; pointer. 8 bytes of local stack
        ; frame are allocated.
...
unlk  a5      ; a5, the old frame pointer, and the
        ; old SP are restored.
```

Since the LINK and UNLK instructions maintain both the frame pointer and the stack pointer, the following code segment is perfectly legal:

```
link  a5,#-4

movem.l d0-a4,-(sp)
pea (500).w
move.l d2,-(sp)
bsr.b Routine

unlk  a5
rts
```

#### FORMAT

For Word size:

```
~~~~~
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0| 1| 0| 0| 1| 1| 1| 0| 0| 1| 0| 1| 0| REGISTER |
|-----|
|                                     16 BITS OFFSET (d)                                     |
|-----
```

For Long size:

```
~~~~~
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0| 1| 0| 0| 1| 0| 0| 0| 0| 0| 0| 0| 0| 1| REGISTER |
|-----|
|          32 BITS OFFSET (d) (16 bits of lower weight)          |
|-----|
|          32 BITS OFFSET (d) (16 bits of upper weight)         |
|-----
```

"REGISTER" indicates the number of address register. OFFSET is a signed value, generally negative, which enables to move the stack pointer.

#### RESULT

None.

SEE ALSO  
UNLK

## 1.87 Logical Shift Left and Logical Shift Right

### NAME

LSL, LSR -- Logical shift left and logical shift right

### SYNOPSIS

LSd Dx,Dy  
LSd #<data>,Dy  
LSd <ea>  
where d is direction, L or R

Size = (Byte, Word, Long)

### FUNCTION

Shift the bits of the operand in the specified direction.  
The carry bit set set to the last bit shifted out of the operand.  
The shift count for the shifting of a register may be specified in two different ways:

1. Immediate - the shift count is specified in the instruction (shift range 1-8).
2. Register - the shift count is contained in a data register specified in the instruction (shift count mod 64)

For a register, the size may be byte, word, or long, but for a memory location, the size must be a word. The shift count is also restricted to one for a memory location.

```
LSL:          <--
             C <----- OPERAND <---- 0
               |
               | (V = 0)
             X <-----'
```

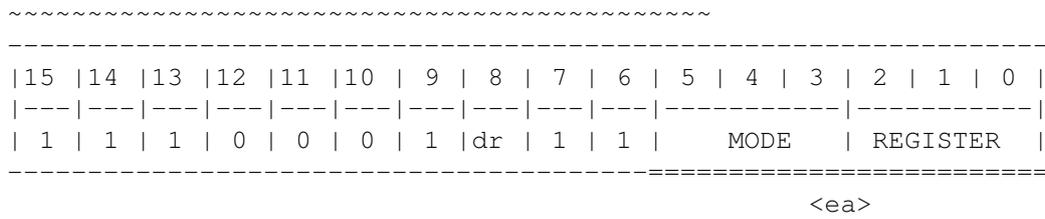
```
LSR:
             -->
0 ----> OPERAND -----> C
               |
               | \----> X
```

### FORMAT

In the case of the shifting of a register:

```
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1| 1| 1| 0| NUMBER/ |dr| SIZE |i/r| 0| 1| REGISTER |
|  |  |  |  | REGISTER |  |  |  |  |  |  |
-----
```

In the case of the shifting of a memory area:



NUMBER/REGISTER

Specifies number of shifting or number of register which contents the number of shifting.

If i/r = 0, number of shifting is specified in the instruction as immediate data

If i/r = 1, it's specified in the data register.

If dr = 0, right shifting

If dr = 1, left shifting

SIZE

00->one Byte operation

01->one Word operation

10->one Long operation

REGISTER

For a register shifting:

Indicates the number of data register on which shifting is applied.

For a memory shifting:

<ea> indicates operand which should be shifted.

Only addressing modes relatives to memory are allowed:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N <sup>textdegree</sup> reg. An	(d16,PC)	-	-
(An)+	011	N <sup>textdegree</sup> reg. An	(d8,PC,Xi)	-	-
-(An)	100	N <sup>textdegree</sup> reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N <sup>textdegree</sup> reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N <sup>textdegree</sup> reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N <sup>textdegree</sup> reg. An	#data	-	-
([bd,An,Xi]od)	110	N <sup>textdegree</sup> reg. An			
([bd,An],Xi,od)	110	N <sup>textdegree</sup> reg. An			

RESULT

- X - Set according to the last bit shifted out of the operand.
- N - Set if the result is negative. Cleared otherwise.
- Z - Set if the result is zero. Cleared otherwise.
- V - Always cleared
- C - Set according to the last bit shifted out of the operand.

SEE ALSO

ASL, ASR ROL, ROR

## 1.88 Move Source -> Destination

NAME

MOVE -- Source -> Destination

SYNOPSIS

MOVE <ea>, <ea>

Size = (Byte, Word, Long)

FUNCTION

Move the content of the source to the destination location. The data is examined as it is moved, and the condition codes set accordingly.

FORMAT



REGISTER

Destination <ea> specifies destination operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-   -		Abs.L	111   001	
(An)	010	N\textdegree{} reg. An	(d16,PC)	-   -	
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-   -	
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-   -	
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-   -	
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-   -	
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-   -	
([bd,An,Xi]od)	110	N\textdegree{} reg. An			

```
|-----|
| ([bd,An],Xi,od) |110 |N\textdegree{} reg. An|
|-----|
```

Source <ea> specifies source operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An *	001	N\textdegree{} reg. An	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	111	011
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	111	100
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

\* Word or Long only.

#### SIZE

01->Byte  
11->Word  
10->Long

#### RESULT

X - Not affected.  
N - Set if the result is negative. Cleared otherwise.  
Z - Set if the result is zero. Cleared otherwise.  
V - Always cleared.  
C - Always cleared.

#### SEE ALSO

MOVEA

## 1.89 Move Address Source -> Destination

#### NAME

MOVEA -- Source -> Destination

#### SYNOPSIS

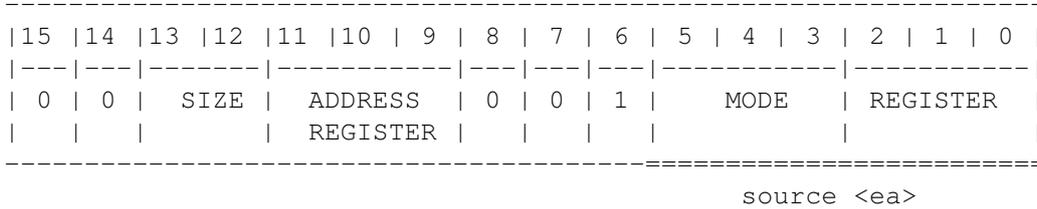
MOVEA <ea>,An

Size = (Word, Long)

FUNCTION

Move the contents of the source to the destination address register. Word sized operands are sign extended to 32 bits before the operation is done.

FORMAT



REGISTER

"ADDRESS REGISTER" specifies the number of destination address register.

Source <ea> specifies source operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N <sup>textdegree</sup> reg. Dn	Abs.W	111	000
An *	001	N <sup>textdegree</sup> reg. An	Abs.L	111	001
(An)	010	N <sup>textdegree</sup> reg. An	(d16,PC)	111	010
(An)+	011	N <sup>textdegree</sup> reg. An	(d8,PC,Xi)	111	011
-(An)	100	N <sup>textdegree</sup> reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N <sup>textdegree</sup> reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N <sup>textdegree</sup> reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N <sup>textdegree</sup> reg. An	#data	111	100
([bd,An,Xi]od)	110	N <sup>textdegree</sup> reg. An			
([bd,An],Xi,od)	110	N <sup>textdegree</sup> reg. An			

\* Word or Long only.

SIZE

11->Word, 32 bits of address register are altered by sign extension.  
 10->Long

RESULT

None.

SEE ALSO

MOVE LEA

## 1.90 CCR -> Destination

### NAME

MOVE from CCR -- CCR -> Destination

### SYNOPSIS

MOVE CCR,<ea>

Size = (Word)

### FUNCTION

The content of the status register is moved to the destination location. The source operand is a word, but only the low order byte contains the condition codes. The high order byte is set to all zeros.

### FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|-----|-----|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |   MODE   | REGISTER |
-----
                                     <ea>

```

### REGISTER

<ea> specifies destination operand, addressing modes allowed are:

```

-----
|Addressing Mode|Mode| Register | |Addressing Mode|Mode|Register|
|-----|-----|-----| |-----|-----|-----|
|      Dn      |000|N\textdegree{} reg. Dn| |   Abs.W     |111| 000 |
|-----|-----|-----| |-----|-----|-----|
|      An      | - | - | |   Abs.L     |111| 001 |
|-----|-----|-----| |-----|-----|-----|
|      (An)     |010|N\textdegree{} reg. An| | (d16,PC)   | - | - |
|-----|-----|-----| |-----|-----|-----|
|      (An)+    |011|N\textdegree{} reg. An| | (d8,PC,Xi) | - | - |
|-----|-----|-----| |-----|-----|-----|
|      -(An)    |100|N\textdegree{} reg. An| | (bd,PC,Xi) | - | - |
|-----|-----|-----| |-----|-----|-----|
|      (d16,An) |101|N\textdegree{} reg. An| | ([bd,PC,Xi],od)| - | - |
|-----|-----|-----| |-----|-----|-----|
|      (d8,An,Xi) |110|N\textdegree{} reg. An| | ([bd,PC],Xi,od)| - | - |
|-----|-----|-----| |-----|-----|-----|
|      (bd,An,Xi) |110|N\textdegree{} reg. An| | #data      | - | - |
|-----|-----|-----| |-----|-----|-----|
| ([bd,An,Xi]od) |110|N\textdegree{} reg. An|
|-----|-----|-----|
| ([bd,An],Xi,od)|110|N\textdegree{} reg. An|
-----

```

### RESULT

None.

### SEE ALSO

Move To CCR

## 1.91 Source -> CCR

### NAME

MOVE to CCR -- Source -> CCR

### SYNOPSIS

MOVE <ea>,CCR

Size = (Word)

### FUNCTION

The content of the source operand is moved to the condition codes. The source operand is a word, but only the low order byte is used to update the condition codes. The high order byte is ignored.

### FORMAT

```

-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|-----|-----|
| 0| 1| 0| 0| 0| 1| 0| 0| 1| 1|   MODE   | REGISTER |
=====
                                     <ea>

```

### REGISTER

<ea> specifies source operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N <sup>textdegree</sup> reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N <sup>textdegree</sup> reg. An	(d16,PC)	111	010
(An)+	011	N <sup>textdegree</sup> reg. An	(d8,PC,Xi)	111	011
-(An)	100	N <sup>textdegree</sup> reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N <sup>textdegree</sup> reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N <sup>textdegree</sup> reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N <sup>textdegree</sup> reg. An	#data	111	100
([bd,An,Xi]od)	110	N <sup>textdegree</sup> reg. An			
([bd,An],Xi,od)	110	N <sup>textdegree</sup> reg. An			

### RESULT

X - Set the same as bit 4 of the source operand.  
 N - Set the same as bit 3 of the source operand.  
 Z - Set the same as bit 2 of the source operand.  
 V - Set the same as bit 1 of the source operand.  
 C - Set the same as bit 0 of the source operand.

SEE ALSO

Move From CCR

## 1.92 Move from SR (privileged)

NAME

MOVE from SR -- Move from status register (privileged)

SYNOPSIS

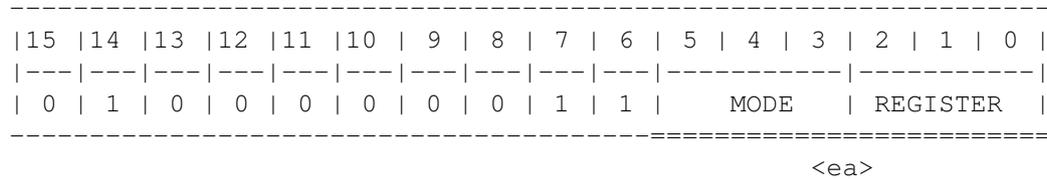
MOVE SR,<ea>

Size = (Word)

FUNCTION

The content of the status register is moved to the destination location. The operand size is a word.

FORMAT



REGISTER

<ea> specifies destination operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-   -		Abs.L	111   001	
(An)	010	N\textdegree{} reg. An	(d16,PC)	-   -	
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-   -	
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-   -	
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-   -	
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-   -	
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-   -	
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

RESULT

None.

SEE ALSO

Move To SR

### 1.93 Move to SR (PRIVILEGED)

NAME

MOVE to SR -- Move to status register (PRIVILEGED)

SYNOPSIS

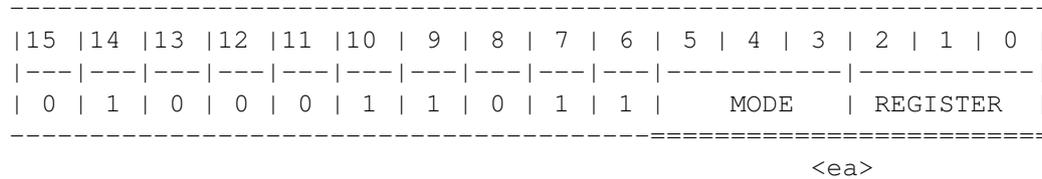
MOVE <ea>,SR

Size = (Word)

FUNCTION

The content of the source operand is moved to the status register. The source operand size is a word and all bits of the status register are affected.

FORMAT



REGISTER

<ea> specifies source operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-   -		Abs.L	111   001	
(An)	010	N\textdegree{} reg. An	(d16,PC)	111   010	
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	111   011	
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111   011	
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111   011	
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111   011	
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	111   100	
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

RESULT

- X - Set the same as bit 4 of the source operand.
- N - Set the same as bit 3 of the source operand.

Z - Set the same as bit 2 of the source operand.  
 V - Set the same as bit 1 of the source operand.  
 C - Set the same as bit 0 of the source operand.

SEE ALSO

MOVE to CCR MOVE from SR

## 1.94 Move to/from USP (privileged)

NAME

MOVE USP -- Move to/from user stack pointer (privileged)

SYNOPSIS

MOVE USP,An  
 MOVE An,USP

Size = (Long)

FUNCTION

The contents of the user stack pointer are transferred either to or from the specified address register.

FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |dr | REGISTER |
-----
```

"REGISTER" indicates the number of address register.

dr specifies move direction:

0->An to USP  
 1->USP to An

RESULT

None.

SEE ALSO

MOVE from CCR MOVE to CCR  
 MOVE from SR MOVE to SR

## 1.95 Move to/from control register

NAME

MOVEC -- Move to/from control register

SYNOPSIS

MOVEC Rc,Rn  
 MOVEC Rn,Rc

Size = (Long)

## FUNCTION

Copy the contents of the specified control register to the specified general register or copy from the general register to the control register. This is always a 32-bit transfer even though the control register may be implemented with fewer bits.

## FORMAT

```

-----
|15|14|13|12|11|10|9|8|7|6|5|4|3|2|1|0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|0|1|0|0|1|1|1|0|0|1|1|1|1|0|1|dr|
|---|-----|-----|-----|-----|-----|-----|-----|
|A/D| REGISTER | CONTROL REGISTER |
-----

```

A/D indicates type of Rn register:

0->Rn=Dn

1->Rn=An

"REGISTER" indicates the number of Rn register.

dr specifies direction of move:

0->Rc to Rn.

1->Rn to Rc.

"CONTROL REGISTER" specifies one of the control registers:

Hexa value	Control Register
~~~~~	~~~~~
000	SFC
001	DFC
002	CACR
800	USP
801	VBR
802	CAAR
803	MSP
804	ISP

All others hexa codes generates an exception: illegal instruction.

## RESULT

None.

## SEE ALSO

MOVE to/from USP

## 1.96 MOVE Multiple registers

## NAME

MOVEM -- Move multiple registers

## SYNOPSIS

MOVEM <register list>, <ea>

MOVEM <ea>, <register list>

Size = (Word, Long)

FUNCTION

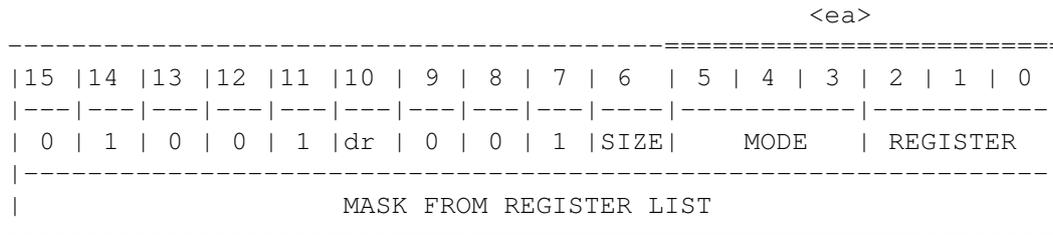
Registers in the register list are either moved to or fetched from consecutive memory locations at the specified address. Data can be either word or long word, but if the register list is destination and the size is word, each register is filled with the source word sign extended to 32-bits.

Also, in the case that the register list is the destination, register indirect with predecrement is not a valid source mode. If the register list is the source, then the destination may not be register indirect with postincrement.

```
MOVEM.L D0/D1/A0, (A2)+    ;invalid
MOVEM.W -(A1), D5/D7/A4   ;invalid
```

The register list is accessed with D0 first through D7, followed by A0 through A7.

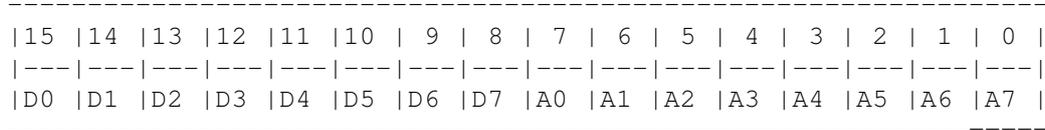
FORMAT



dr specifies move direction:  
 0->registers to memory  
 1->memory to registers

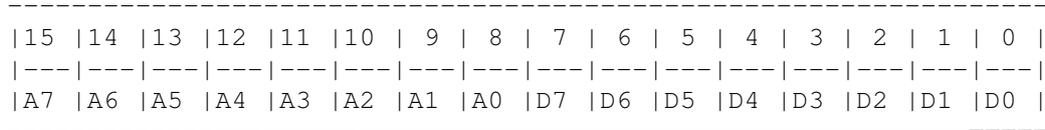
MASK FROM REGISTER LIST specifies registers which must be moved and indicates their move order.

For pre-decrementing, mask has the following format:



First register to be moved-----'

For post-incrementing, mask has the following format:



First register to be moved-----'

## SIZE

0-&gt;16 bits.

1-&gt;32 bits.

## REGISTER

&lt;ea&gt; specifies memory address of move.

Move from registers to memory, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111   000	
An	-	-	Abs.L	111   001	
(An)	010	N\textdegree{} reg. An	(d16,PC)	-   -	
(An)+	-	-	(d8,PC,Xi)	-   -	
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-   -	
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-   -	
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-   -	
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-   -	
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

Move from memory to registers, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111   000	
An	-	-	Abs.L	111   001	
(An)	010	N\textdegree{} reg. An	(d16,PC)	111   010	
(An)+	-	-	(d8,PC,Xi)	111   011	
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111   011	
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111   011	
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111   011	
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-   -	
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

RESULT  
None.

SEE ALSO  
MOVE MOVEA MOVEP

## 1.97 MOVE Peripheral data

NAME  
MOVEP -- Move peripheral data

SYNOPSIS  
MOVEP Dx, (d,Ay)  
MOVEP (d,Ay),Dx

Size = (Word, Long)

FUNCTION  
Data is transferred between a data register and every other byte of memory at the selected address. Transfer is made between a data register and alternated bytes of memory at the selected address, must be specified in indirect mode to An with a 16 bits displacement. This instruction is of use with 8 bits peripheral programming.

Example:

```
~~~~~
LEA port0,A0 ; A0 -> $FFFFFFFFFFFFFFFF
MOVEQ #0,D0
MOVEP.L D0, (0,A0) ; A0 -> $FF00FF00FF00FF00
MOVE.L #$55554444,D0
MOVEP.L D0, (1,A0) ; A0 -> $FF55FF55FF44FF44
```

FORMAT

```
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|-----|-----|---|---|---|-----|
| 0| 0| 0| 0|Dx REGISTER| OP-MODE | 0| 0| 1|Ay REGISTER|
|-----|
|                                     16 BITS OFFSET                                     |
|-----
```

OP-MODE  
100->16 bits move, memory to register  
101->32 bits move, memory to register  
110->16 bits move, register to memory  
111->32 bits move, register to memory

REGISTER  
Dx register specifies the number of data register.  
Ay register specifies the number of address register which takes place in indirect addressing with displacement.

RESULT  
None.

SEE ALSO  
 MOVEM

## 1.98 MOVE signed 8-bit data Quick

NAME

MOVEQ -- Move signed 8-bit data quick

SYNOPSIS

MOVEQ #<data:8>,Dn

Size = (Long)

FUNCTION

Move signed 8-bit data to the specified data register.  
 The specified data is sign extended to 32-bits before  
 it is moved to the register.

FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|-----|---|-----|
| 0 | 1 | 1 | 1 | REGISTER | 0 |           IMMEDIATE DATA |
-----
```

"REGISTER" is one of the 8 datas registers.

RESULT

X - Not affected.  
 N - Set if the result is negative. Cleared otherwise.  
 Z - Set if the result is zero. Cleared otherwise.  
 V - Always cleared.  
 C - Always cleared.

SEE ALSO

MOVE MOVEA LEA

## 1.99 MOVE address Space (PRIVILEGED)

NAME

MOVES -- Move address space (PRIVILEGED)

SYNOPSIS

MOVES Rn,<ea> (68010+)  
 MOVES <ea>,Rn (68010+)

Size = (Byte, Word, Long)

FUNCTION

Moves contents (Byte, Word, Long) of register Rn to the addressed  
 space by effective address in the addressable space specified by DFC.

If destination is an address register, there's extension of operand sign to 32 bits.

FORMAT

<ea>															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
0	0	0	0	1	1	1	0	SIZE		MODE		REGISTER			
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
A/D	Rn	REGISTER	dr	0	0	0	0	0	0	0	0	0	0	0	0

A/D indicates type of Rn register:

0->Rn=Dn

1->Rn=An

"REGISTER" indicates the number of Rn register.

dr specifies direction of move:

0->memory to register.

1->register to memory.

SIZE

00->Byte

01->Word

10->Long

REGISTER

<ea> specifies place of source or destination, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

RESULT

None.

#### SEE ALSO

MOVE from CCR MOVE to CCR  
 MOVE from SR MOVE to SR  
 MOVE to/from USP MOVEM  
 MOVEC MOVEP

## 1.100 Signed and Unsigned MULtipl

#### NAME

MULS -- Signed multiply  
 MULU -- Unsigned multiply

#### SYNOPSIS

MULS.W <ea>,Dn 16\*16->32  
 MULS.L <ea>,Dn 32\*32->32 68020+  
 MULS.L <ea>,Dh:Dl 32\*32->64 68020+  
  
 MULU.W <ea>,Dn 16\*16->32  
 MULU.L <ea>,Dn 32\*32->32 68020+  
 MULU.L <ea>,Dh:Dl 32\*32->64 68020+

Size = (Word)  
 Size = (Word, Long) 68020+

#### FUNCTION

Multiply two signed (MULS) or unsigned (MULU) integers to produce either a signed or unsigned, respectively, result.

This instruction has three forms. They are basically word, long word, and quad word. The first version is the only one available on a processor lower than a 68020. It will multiply two 16-bit integers and produce a 32-bit result. The second will multiply two 32-bit integers and produce a 32-bit result.

The third form needs some special consideration. It will multiply two 32-bit integers, specified by <ea> and Dl, the result is (sign) extended to 64-bits with the low order 32 being placed in Dl and the high order 32 being placed in Dh.

#### FORMAT

In the case of MULS.W:

```

~~~~~
<ea>
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|-----|---|---|---|-----|
| 1| 1| 0| 0| REGISTER | 1| 1| 1|   MODE   | REGISTER |
-----

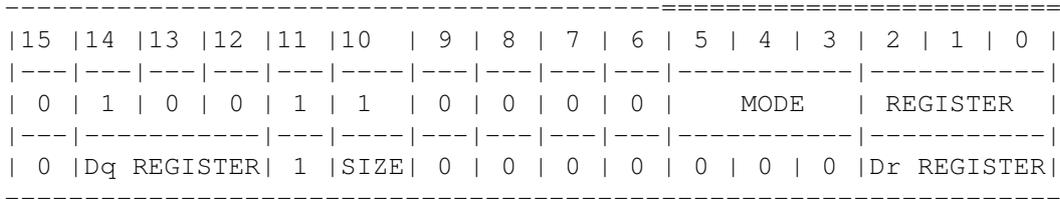
```

In the case of MULS.L:

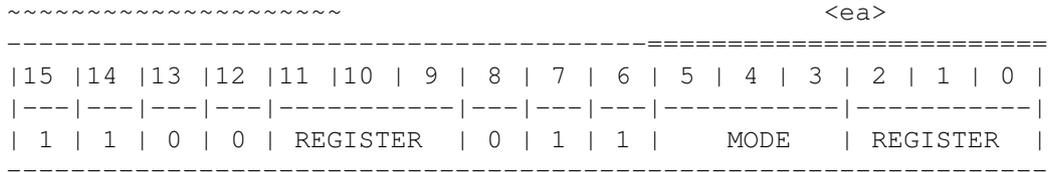
```

~~~~~
<ea>

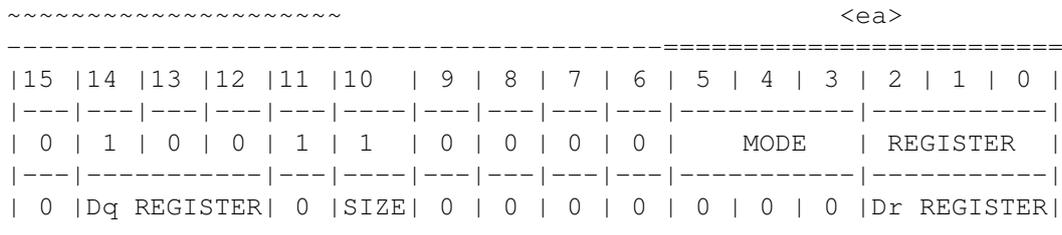
```



In the case of MULU.W:



In the case of MULU.L:



"REGISTER" indicates the number of data register.

"Dq REGISTER" indicates the number of data register for destination operand. This register first contains 32 bits of low weight of dividend, and after the value of quotient on 32 bits.

"SIZE" specifies if dividend is on 32 or 64 bits:  
 0-> 32 bits dividend placed in Dq.  
 1-> 64 bits dividend placed in Dr:Dq.

"Dr REGISTER" indicates the number of data register for destination operand. This register first contains 32 bits of upper weight of dividend if "SIZE" = 1, and after the value of rest on 32 bits.

If Dr and Dq represents the same register, only quotient on 32 bits is put in Dq.

<ea> field specifies source operand, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N <sup>textdegree</sup> reg. Dn	Abs.W	111	000
An	-   -	Abs.L	111	001	
(An)	010	N <sup>textdegree</sup> reg. An	(d16,PC)	111	010
(An)+	011	N <sup>textdegree</sup> reg. An	(d8,PC,Xi)	111	011
-(An)	100	N <sup>textdegree</sup> reg. An	(bd,PC,Xi)	111	011



C - Set if a borrow was generated, cleared otherwise.

SEE ALSO

NEG NEGX

## 1.102 neg

NAME

NEG -- Negate

SYNOPSIS

NEG <ea>

Size = (Byte, Word, Long)

FUNCTION

The operand specified by <ea> is subtracted from zero. The result is stored in <ea>.

FORMAT

```

-----
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---|
| 0  | 1  | 0  | 0  | 0  | 1  | 0  | 0  | SIZE | MODE | REGISTER |
-----=====
                                     <ea>

```

SIZE

00->Byte.

01->Word.

10->Long.

REGISTER

<ea> specifies destination operand, addressing modes allowed are:

```

-----
|Addressing Mode|Mode| Register | |Addressing Mode|Mode|Register|
|-----|-----|-----| |-----|-----|-----|
|      Dn      |000| N\textdegree{} reg. Dn| | Abs.W      |111| 000 |
|-----|-----|-----| |-----|-----|-----|
|      An      | - | - | | Abs.L      |111| 001 |
|-----|-----|-----| |-----|-----|-----|
|     (An)     |010| N\textdegree{} reg. An| | (d16,PC)  | - | - |
|-----|-----|-----| |-----|-----|-----|
|     (An)+    |011| N\textdegree{} reg. An| | (d8,PC,Xi)| - | - |
|-----|-----|-----| |-----|-----|-----|
|    -(An)    |100| N\textdegree{} reg. An| | (bd,PC,Xi)| - | - |
|-----|-----|-----| |-----|-----|-----|
|  (d16,An)   |101| N\textdegree{} reg. An| | ([bd,PC,Xi],od)| - | - |
|-----|-----|-----| |-----|-----|-----|
|  (d8,An,Xi) |110| N\textdegree{} reg. An| | ([bd,PC],Xi,od)| - | - |
|-----|-----|-----| |-----|-----|-----|
|  (bd,An,Xi) |110| N\textdegree{} reg. An| | #data     | - | - |
|-----|-----|-----| |-----|-----|-----|
| ([bd,An,Xi]od)|110| N\textdegree{} reg. An|
|-----|

```

```
|([bd,An],Xi,od)|110 |N\textdegree{} reg. An|
-----
```

## RESULT

X - Set the same as the carry bit.  
 N - Set if the result is negative, otherwise cleared.  
 Z - Set if the result is zero, otherwise cleared.  
 V - Set if overflow, otherwise cleared.  
 C - Cleared if the result is zero, otherwise set.

## SEE ALSO

NBCD NEGX

## 1.103 NEGate with eXtend

## NAME

NEGX -- Negate with extend

## SYNOPSIS

NEGX <ea>

Size = (Byte, Word, Long)

## FUNCTION

Perform an operation similar to a NEG, with the exception that the operand and the extend bit are both subtracted from zero. The result then is being placed in the given <ea>.

As with ADDX, SUBX, ABCD, SBCD, and NBCD, it can be useful to set the zero flag before performing this operation so that multi precision operations can be tested for zero.

## FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|-----|-----|-----|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | SIZE | MODE | REGISTER |
-----=====
```

<ea>

## SIZE

00->Byte.  
 01->Word.  
 10->Long.

## REGISTER

<ea> specifies destination operand, addressing modes allowed are:

```
-----
|Addressing Mode|Mode| Register | |Addressing Mode|Mode|Register|
|-----|-----|-----| |-----|-----|-----|
|          Dn          |000 |N\textdegree{} reg. Dn| |      Abs.W          |111 | 000 |
|-----|-----|-----| |-----|-----|-----|
|          An          | - | - | |      Abs.L          |111 | 001 |
-----
```

(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-	
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-	
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-	
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-	
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-	
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-	
([bd,An,Xi]od)	110	N\textdegree{} reg. An				
([bd,An],Xi,od)	110	N\textdegree{} reg. An				

#### RESULT

- X - Set the same as the carry bit.
- N - Set if the result is negative, otherwise cleared.
- Z - Cleared if the result is non-zero, otherwise unchanged.
- V - Set if an overflow is generated, cleared otherwise.
- C - Set if a borrow is generated, otherwise cleared.

#### SEE ALSO

NEG NBCD ADDX SUBX

## 1.104 No Operation

#### NAME

NOP -- No operation

#### SYNOPSIS

NOP

#### FUNCTION

Nothing happens! This instruction will basically wait until all pending bus activity is completed. This allows synchronization of the pipeline and prevents instruction overlap.

#### FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1	

#### RESULT

None.

#### SEE ALSO

## 1.105 Logical complement

### NAME

NOT -- Logical complement

### SYNOPSIS

NOT <ea>

Size = (Byte, Word, Long)

### FUNCTION

All bits of the specified operand are inverted and placed back in the operand.

### FORMAT

```
-----
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---|
| 0  | 1  | 0  | 0  | 0  | 1  | 1  | 0  | SIZE |     | MODE  | REGISTER |
=====
```

<ea>

### SIZE

00->Byte.  
01->Word.  
10->Long.

### REGISTER

<ea> specifies destination operand, addressing modes allowed are:

```
-----
| Addressing Mode | Mode | Register | | Addressing Mode | Mode | Register |
|-----|-----|-----| |-----|-----|-----|
|      Dn         | 000 | N\textdegree{} reg. Dn | | Abs.W         | 111 | 000      |
|-----|-----|-----| |-----|-----|-----|
|      An         | -   | -           | | Abs.L         | 111 | 001      |
|-----|-----|-----| |-----|-----|-----|
|      (An)       | 010 | N\textdegree{} reg. An | | (d16,PC)      | -   | -        |
|-----|-----|-----| |-----|-----|-----|
|      (An)+      | 011 | N\textdegree{} reg. An | | (d8,PC,Xi)    | -   | -        |
|-----|-----|-----| |-----|-----|-----|
|      -(An)      | 100 | N\textdegree{} reg. An | | (bd,PC,Xi)    | -   | -        |
|-----|-----|-----| |-----|-----|-----|
|      (d16,An)   | 101 | N\textdegree{} reg. An | | ([bd,PC,Xi],od) | -   | -        |
|-----|-----|-----| |-----|-----|-----|
|      (d8,An,Xi) | 110 | N\textdegree{} reg. An | | ([bd,PC],Xi,od) | -   | -        |
|-----|-----|-----| |-----|-----|-----|
|      (bd,An,Xi) | 110 | N\textdegree{} reg. An | | #data         | -   | -        |
|-----|-----|-----| |-----|-----|-----|
| ([bd,An,Xi]od) | 110 | N\textdegree{} reg. An |
|-----|-----|-----|
| ([bd,An],Xi,od) | 110 | N\textdegree{} reg. An |
-----
```

### RESULT

X - Not affected.  
N - Set if the result is negative, otherwise cleared.

Z - Set if the result is zero, otherwise cleared.  
 V - Always cleared.  
 C - Always cleared.

SEE ALSO  
 NEG

## 1.106 Logical OR

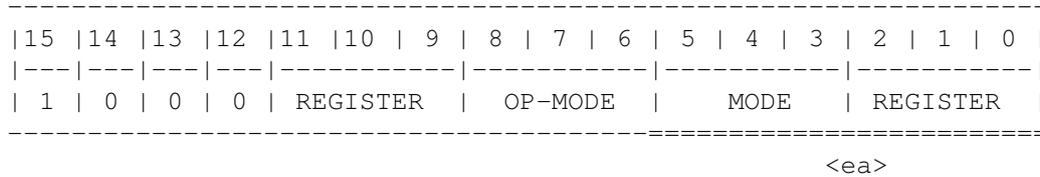
NAME  
 OR -- Logical OR

SYNOPSIS  
 OR <ea>,Dn  
 OR Dn,<ea>

Size = (Byte, Word, Long)

FUNCTION  
 Performs an OR operation on the destination operand with the source operand.

FORMAT



OP-MODE

Byte	Word	Long	Operation
000	001	010	<ea> OR <Dn> --> <Dn>
100	101	110	<Dn> OR <ea> --> <ea>

REGISTER

If <ea> specifies destination operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111   000	
An	-	-	Abs.L	111   001	
(An)	010	N\textdegree{} reg. An	(d16,PC)	-   -	
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-   -	
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-   -	
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-   -	
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-   -	

```

| (bd,An,Xi) |110 |N\textdegree{} reg. An| | #data | - | - |
|-----|-----|
| ([bd,An,Xi]od) |110 |N\textdegree{} reg. An|
|-----|
| ([bd,An],Xi,od) |110 |N\textdegree{} reg. An|
|-----|

```

If <ea> specifies source operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	111	011
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	111	100
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

#### RESULT

X - Not Affected  
N - Set to the value of the most significant bit.  
Z - Set if the result is zero.  
V - Always cleared  
C - Always cleared

#### SEE ALSO

ORI BSET

## 1.107 Logical OR Immediate

#### NAME

ORI -- Logical OR immediate

#### SYNOPSIS

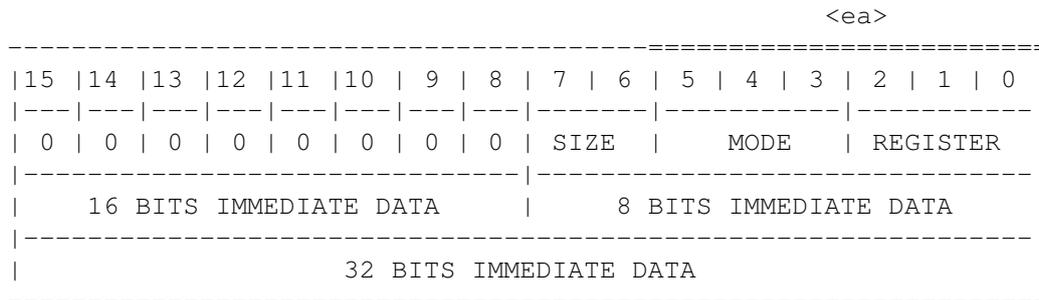
ORI #<data>,<ea>

Size = (Byte, Word, Long)

#### FUNCTION

Performs an inclusive OR operation on the destination operand with the source operand.

## FORMAT



## SIZE

00->Byte.  
01->Word.  
10->Long.

## REGISTER

<ea> specifies destination operand, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

## RESULT

X - Not Affected  
N - Set to the value of the most significant bit.  
Z - Set if the result is zero.  
V - Always cleared  
C - Always cleared

## SEE ALSO

OR ORI to CCR ORI to SR  
BSET

## 1.108 Logical OR immediate to CCR

### NAME

ORI to CCR -- Logical OR immediate to the condition code register

### SYNOPSIS

ORI #<data>,CCR

Size = (Byte)

### FUNCTION

Performs an OR operation on the condition codes register with the source operand.

### FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |           8 BITS IMMEDIATE DATA |
-----

```

### RESULT

X - Set if bit 4 of the source is set, cleared otherwise.  
 N - Set if bit 3 of the source is set, cleared otherwise.  
 Z - Set if bit 2 of the source is set, cleared otherwise.  
 V - Set if bit 1 of the source is set, cleared otherwise.  
 C - Set if bit 0 of the source is set, cleared otherwise.

### SEE ALSO

OR ORI ORI to SR

## 1.109 Logical OR immediated to SR (PRIVILEGED)

### NAME

ORI to SR -- Logical OR immediated to the status register (PRIVILEGED)

### SYNOPSIS

ORI #<data>,SR

Size = (Word)

### FUNCTION

Performs an OR operation on the status register with the source operand, and leaves the result in the status register.

### FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|-----|-----|-----|

```

| 16 BITS IMMEDIATE DATA |

-----

## RESULT

X - Set if bit 4 of the source is set, cleared otherwise.  
 N - Set if bit 3 of the source is set, cleared otherwise.  
 Z - Set if bit 2 of the source is set, cleared otherwise.  
 V - Set if bit 1 of the source is set, cleared otherwise.  
 C - Set if bit 0 of the source is set, cleared otherwise.

## SEE ALSO

OR ORI ORI to CCR

## 1.110 PACK binary coded decimal

## NAME

PACK -- Pack binary coded decimal (68020+)

## SYNOPSIS

PACK -(Ax), -(Ay), #<adjustment>  
 PACK Dx, Dy, #<adjustment>

Size = (Byte, Long)

## FUNCTION

Convert byte-per-digit unpacked BCD to packed two-digit-per-byte BCD.

When operand is in a data register, 16 bits adjustment is added to source operand (16 bits). Then, bits 8 to 11 and 0 to 3 are packed and placed in the bits 0 to 7 of destination register. Others bits of this register are not altered.

When operand is in memory, there's a research by pre-decrementing of the two bytes placed at given address. The two bytes are linked together, then adjustment is added, and the compacted result is stored at destination's place.

## FORMAT

```

-----
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---|
| 1  | 0  | 0  | 0  |   | Dy/Ay | 1  | 0  | 1  | 0  | 0  | R/M |   | Dx/Ax |
| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---|
|                                     16 BITS ADJUSTMENT
-----

```

R/M = 0 -> Direct addressing by data register.

R/M = 1 -> Addressing by pre-decrementing.

Register Dy/Ay specifies destination register.

Register Dx/Ax specifies source register.

"16 BITS ADJUSTMENT" is an immediate value added to source operand.

## RESULT

None.

SEE ALSO  
UNPK

## 1.111 Push Effective Address

### NAME

PEA -- Push effective address

### SYNOPSIS

PEA <ea>

Size = (Long)

### FUNCTION

Effective address is stored to stack. Stack is decreased.

### FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |   MODE   | REGISTER |
-----
                                     <ea>
```

### REGISTER

<ea> specifies destination operand, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	-	-	(d8,PC,Xi)	111	011
-(An)	-	-	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

### RESULT

None.

SEE ALSO

LEA

## 1.112 Invalidate one or several entries in the ATC (PRIVILEGED)

NAME

PFLUSH -- Invalidate one or several entries in the ATC (PRIVILEGED)

SYNOPSIS

```
PFLUSHA
PFLUSH <FC>, #<FC validation>
PFLUSH <FC>, #<FC validation>, <ea>
```

No size specs.

FUNCTION

Those three instructions are used to invalidate one or several entries of the ATC (PMMU cache), i.e. to set the validation bits followings to those zero entries.

PFLUSHA invalidate all the 22 entries of cache.

PFLUSH <FC>, #<FC validation> invalidate entry which follows mentioned Function Codes.

PFLUSH <FC>, #<FC validation>, <ea> invalidate entry which address is specified in the destination, of course in taking care of Function Codes.

FC validation bits allow to take care of 3 FC bits else only some of these bits.

The status register of the PMMU, MMUSR, isn't affected by this instruction.

<FC> operand can be mentioned:

- in immediate.
- by the three lower bits of a data register.
- by the register SFC or DFC.

FORMAT

														<ea>		
-----																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	
1	1	1	1	0	0	0	0	0	0	MODE			REGISTER			
---	---	---	---	---	---	---	---	---	---	-----			-----			
0	0	1	MODE			0	0	FC			FC					
								VALIDATION								
-----																

MODE field indicates the type of PFLUSH:

001->invalidation of all entries.

100->invalidation by Function Codes.

110->invalidation by the Function Codes and <ea>.

FC VALIDATION field indicates the FC bits to take care of.

FC field indicates value of Function Codes of the entry to invalidate.

10XXX The Function Codes are XXX.

01DDD The Function Codes are the bits 2 to 0 of a DDD data register.

0000 The Function Codes are specified in SFC.

0001 The Function Codes are specified in DFC.

#### REGISTER

<ea> specifies address to invalidate, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	-	-	(d8,PC,Xi)	-	-
-(An)	-	-	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

#### RESULT

Not affected.

#### SEE ALSO

PLOAD

## 1.113 LOAD of an entry in the ATC (PRIVILEGED)

#### NAME

PLOAD -- Load of an entry in the ATC (PRIVILEGED)

#### SYNOPSIS

PLOADR <FC>, <ea>

PLOADW <FC>, <ea>

No size specs.

#### FUNCTION

Those instructions are used to load in the ATC (PMMU cache), one entry following to logic specified <ea>. A research in table is made for this cache modification and attributes of final descriptor are updated





```
| 0 | 1 | 0 | MMU-reg | R/W | FD | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
```

For TT0, TT1, registers:

```
~~~~~
PMMU-reg field specifies PMMU register:
010->TT0
011->TT1
```

FD bit: allows or not ATC invalidation:  
 0->ATC invalidated.  
 1->ATC NOT invalidated.

```

                                     <ea>
=====
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |   MODE   | REGISTER |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | MMU-reg | R/W | FD | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
```

For MMUSR register:

```

                                     <ea>
=====
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |   MODE   | REGISTER |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | R/W | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
```

R/W field indicates type of access used for research:  
 0->Memory to PMMU register.  
 1->PMMU register to memory.

REGISTER

<ea> specifies memory address, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N <sup>textdegree</sup> reg. An	(d16,PC)	-	-
(An)+	-	-	(d8,PC,Xi)	-	-
-(An)	-	-	(bd,PC,Xi)	-	-
(d16,An)	101	N <sup>textdegree</sup> reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N <sup>textdegree</sup> reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N <sup>textdegree</sup> reg. An	#data	-	-

```

|-----|-----|
| ([bd,An,Xi]od) |110 |N\textdegree{} reg. An|
|-----|-----|
| ([bd,An],Xi,od) |110 |N\textdegree{} reg. An|
|-----|-----|

```

## RESULT

Not affected.

## SEE ALSO

PTEST

## 1.115 TESTs a logic address (PRIVILEGED)

## NAME

PTEST -- TESTs a logic address (PRIVILEGED)

## SYNOPSIS

```

PTESTR <FC>, <ea>, #<level>
PTESTR <FC>, <ea>, #<level>, An
PTESTW <FC>, <ea>, #<level>
PTESTW <FC>, <ea>, #<level>, An

```

No size specs.

## FUNCTION

This instruction examines the ATC, if level is equal to zero, a research in the translation tables is made, if level is different of zero (1 to 7), then sets MMUSR bits.

This instruction can also store, in an address register An, physical address encountered to last level of its research.

PTESTR or PTESTW version is used to simulate a read or write cycle and like this, according to the informations founds, exactly set MMUSR.

MMUSR bits are set of the following manner:

MMUSR bits	PTEST level 0	PTEST level > 0
B (bit 15)	This bit is set if the bit "Error Bus (B)" of the ATC is set.	This bit is set if a bus error is generated during research in the tables.
L (bit 14)	This bit is cleared	This bit is set if an index overflow a limit during a research.
S (bit 13)	This bit is cleared	This bit is set for indicating a privilege violation: if S bit of one of the descriptors met is set and the FC2 bit mentioned in the instruction is cleared (user access). S isn't defined if the bit I of MMUSR is set.

W (bit 11)	This bit is set if the   bit WP in entry of the   examined ATC is set.   Undefined if I is set.	This bit is set if WP bit of   one of the descriptors   encountered is set. Undefined   if I is set.
I (bit 10)	This bit is set if   required logic address   isn't in the ATC or if   the bit B of this entry   is set.	This bit is set if one of the   descriptors encountered isn't   valid (DT = 0) or if B or L of   MMUSR are set during research.
M (bit 9)	This bit is set if the   bit M of designed entry   is set. Undefined if I   is set.	This bit is set if the encountered   page descriptor has its bit M   set. Undefined if I is set.
T (bit 6)	This bit is set if   mentioned logic address   is part of defined   window by TT0 and/or   TT1.	This bit is cleared.
N (bits 2 to 0)	This field is cleared	This field represents the number   of level accessed during table   research.

<FC> operand can be mentioned:

- in immediate.
- by the three lower bits of a data register.
- by the register SFC or DFC.

#### FORMAT

															<ea>		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
1	1	1	1	0	0	0	0	0	0								
---	---	---	---	---	---	---	---	---	---								
1	0	0															

R/W field indicates type of access used for research:

- 0->write access.
- 1->read access.

FC field indicates value of Function Codes of the address to test

10XXX The Function Codes are XXX.

01DDD The Function Codes are the bits 2 to 0 of a DDD data register.

0000 The Function Codes are specified in SFC.

0001 The Function Codes are specified in DFC.

Bit A specifies address register option:

- 0-> no address register
- 1-> address of last accessed descriptor is put in the register specified by REG.

REG field indicates, if A = 1 the number of address register. Else if A = 0, REG = 0.

LEVEL field indicates the highest logic level to go during research; if test in the ATC, LEVEL = 0.

#### REGISTER

<ea> specifies logic address to test, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	-	-	(d8,PC,Xi)	-	-
-(An)	-	-	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

#### RESULT

Not affected.

#### SEE ALSO

PLOAD

## 1.116 RESET external devices

#### NAME

RESET -- Reset external devices

#### SYNOPSIS

RESET

#### FUNCTION

RESET line is set, then external circuitry is reset.

Processor state isn't modified, except PC, which allows restart of execution to the next instruction.

If processor is NOT in supervisor state, there's generation of exception "privilege violation", vector n\textdegree{}8.

#### FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
-----

```

RESULT  
None.

SEE ALSO  
STOP

## 1.117 RESET external devices

NAME  
RESET -- Reset external devices

SYNOPSIS  
RESET

FUNCTION  
RESET line is set, then external circuitry is reset.  
Processor state isn't modified, except PC, which allows restart of execution to the next instruction.  
If processor is NOT in supervisor state, there's generation of exception "privilege violation", vector n\textdegree{}8.

FORMAT

```

-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
-----

```

RESULT  
None.

SEE ALSO  
STOP

## 1.118 ROTate Left and ROTate Right

NAME  
ROL, ROR -- Rotate left and rotate right

SYNOPSIS  
ROd Dx,Dy  
ROd #<data>,Dy  
ROd <ea>  
where d is direction, L or R

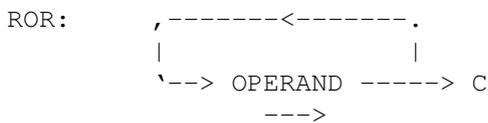
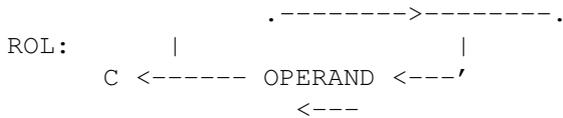
Size = (Byte, Word, Long)

FUNCTION

Rotate the bits of the operand in the specified direction.  
 The rotation count may be specified in two different ways:

1. Immediate - the rotation count is specified in the instruction
2. Register - the rotation count is contained in a data register specified in the instruction

For a register, the size may be byte, word, or long, but for a memory location, the size must be a word. The rotation count is also restricted to one for a memory location.



FORMAT

In the case of the rotating of a register:

```

-----
|15|14|13|12|11|10|9|8|7|6|5|4|3|2|1|0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | NUMBER/ | dr | SIZE | i/r | 1 | 1 | REGISTER |
|   |   |   |   | REGISTER |   |   |   |   |   |   |   |
-----
    
```

In the case of the rotating of a memory area:

```

-----
|15|14|13|12|11|10|9|8|7|6|5|4|3|2|1|0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | dr | 1 | 1 | MODE | REGISTER |
|   |   |   |   |   |   |   |   |   |   |   |   |   |
-----
    
```

<ea>

NUMBER/REGISTER

Specifies number of rotating or number of register which contents the number of rotating.  
 If i/r = 0, number of rotating is specified in the instruction as immediate data  
 If i/r = 1, it's specified in the data register.  
 If dr = 0, right rotating  
 If dr = 1, left rotating

SIZE

00->one Byte operation  
 01->one Word operation  
 10->one Long operation

## REGISTER

For a register rotating:

Indicates the number of data register on which rotating is applied.

For a memory rotating:

<ea> indicates operand which should be rotated.

Only addressing modes relatives to memory are allowed:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

## RESULT

X - Not affected

N - Set if the result is negative. Cleared otherwise.

Z - Set if the result is zero. Cleared otherwise.

V - Always cleared

C - Set according to the last bit shifted out of the operand.

## SEE ALSO

ROXL, ROXR ASL, ASR LSL, LSR

## 1.119 ROTate Left with eXtend and ROTate Right with eXtend

## NAME

ROXL, ROXD -- Rotate left with extend and rotate right with extend

## SYNOPSIS

ROXd Dx,Dy

ROXd #<data>,Dy

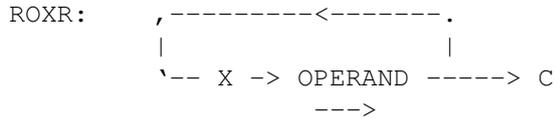
ROXd <ea>

where d is direction, L or R

Size = (Byte, Word, Long)

FUNCTION

A rotation is made on destination operand bits.  
 Rotation uses bit X.



FORMAT

In the case of the rotating of a register:

```

-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1| 1| 1| 0| NUMBER/ |dr| SIZE |i/r| 1| 0| REGISTER |
|   |   |   |   | REGISTER |   |   |   |   |   |   |   |
-----
    
```

In the case of the rotating of a memory area:

```

-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1| 1| 1| 0| 0| 1| 0|dr| 1| 1|   | MODE | REGISTER |
|-----|-----|-----|-----|-----|-----|-----|-----|
-----
    
```

<ea>

NUMBER/REGISTER

Specifies number of rotating or number of register which contents the number of rotating.

If i/r = 0, number of rotating is specified in the instruction as immediate data

If i/r = 1, it's specified in the data register.

If dr = 0, right rotating

If dr = 1, left rotating

SIZE

00->one Byte operation

01->one Word operation

10->one Long operation

REGISTER

For a register rotating:

Indicates the number of data register on which rotating is applied.

For a memory rotating:

<ea> indicates operand which should be rotated.

Only addressing modes relatives to memory are allowed:

```

-----
|Addressing Mode|Mode| Register | |Addressing Mode|Mode|Register|
-----
    
```

Dn	-	-	Abs.W	111		000	
An	-	-	Abs.L	111		001	
(An)	010	N\textdegree{}	reg. An		(d16,PC)		-   -
(An)+	011	N\textdegree{}	reg. An		(d8,PC,Xi)		-   -
-(An)	100	N\textdegree{}	reg. An		(bd,PC,Xi)		-   -
(d16,An)	101	N\textdegree{}	reg. An		([bd,PC,Xi],od)		-   -
(d8,An,Xi)	110	N\textdegree{}	reg. An		([bd,PC],Xi,od)		-   -
(bd,An,Xi)	110	N\textdegree{}	reg. An		#data		-   -
([bd,An,Xi]od)	110	N\textdegree{}	reg. An				
([bd,An],Xi,od)	110	N\textdegree{}	reg. An				

**RESULT**

- X - Set by the last bit out of operand.  
Not changed if rotation is zero.
- N - Set if the result is negative. Cleared otherwise.
- Z - Set if the result is zero. Cleared otherwise.
- V - Always cleared
- C - Set according to the last bit shifted out of the operand.

**SEE ALSO**

ROL, ROR ASL, ASR LSL, LSR

**1.120 ReTurn and Deallocate parameter stack frame****NAME**

RTD -- Return and deallocate parameter stack frame (68010+)

**SYNOPSIS**

RTD #<offset>

**FUNCTION**

PC is subtracted from stack and replace old PC address.  
Then offset is added to SP value.

This instruction is useful to restore reserved space memory of stored arguments at time sub-routine is called.

**FORMAT**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0	
16 BITS OFFSET																

-----  
 "16 BITS OFFSET" is a signed 16 bits value to add to SP.

RESULT  
 None.

SEE ALSO  
 RTS RTE RTM

## 1.121 ReTurn from Exception (PRIVILEGED)

NAME  
 RTE -- Return from exception (PRIVILEGED)

SYNOPSIS  
 RTE

FUNCTION  
 SR and PC are restored by SP. All SR bits are affected.

FORMAT

```
-----
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---| ---|
| 0  | 1  | 0  | 0  | 1  | 1  | 1  | 0  | 0  | 1  | 1  | 1  | 0  | 0  | 1  | 1  |
-----
```

RESULT  
 SR is set following to the restored word taken from SP.

SEE ALSO  
 RTS RTD RTM

## 1.122 ReTurn from process Module

NAME  
 RTM -- Return from process module (68020 ONLY)

SYNOPSIS  
 RTM Rn

FUNCTION  
 Return from a process module called with CALLM.  
 This instruction is 68020 ONLY and is used with, for cooperation  
 with the PMMU MC68851. Be carreful, it's not available on 68030+.

RESULT  
 Don't know!

SEE ALSO  
 RTS RTE RTD

---

## 1.123 ReTurn and Restore CCR

### NAME

RTR -- Return and restore condition code register

### SYNOPSIS

RTR

### FUNCTION

CCR and PC are restored by SP.  
Supervisor byte of SR isn't affected.

### FORMAT

```
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0| 1| 0| 0| 1| 1| 1| 0| 0| 1| 1| 1| 0| 1| 1| 1|
-----
```

### RESULT

SR is set following to the restored word taken from SP.

### SEE ALSO

RTS RTE RTD

## 1.124 ReTurn from Subroutine

### NAME

RTS -- Return from subroutine

### SYNOPSIS

RTS

### FUNCTION

PC is restored by SP.

### FORMAT

```
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0| 1| 0| 0| 1| 1| 1| 0| 0| 1| 1| 1| 0| 1| 0| 1|
-----
```

### RESULT

None.

### SEE ALSO

RTM RTE RTD

## 1.125 Subtract Binary Coded Decimal with extend

## NAME

SBCD -- Subtract binary coded decimal with extend

## SYNOPSIS

SBCD Dy, Dx  
 SBCD -(Ay), -(Ax)

Size = (Byte)

## FUNCTION

Subtracts the source operand to the destination operand along with the extend bit, and stores the result in the destination location.

The subtraction is performed using binary coded decimal arithmetic. The operands, which are packed BCD numbers, can be addressed in two different ways:

1. Data register to data register: The operands are contained in the data registers specified in the instruction.
2. Memory to memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

Normally the Z condition code bit is set via programming before the start of an operation. That allows successful tests for zero results upon completion of multiple-precision operations.

## FORMAT

```

-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|-----|---|---|---|---|---|---|-----|
| 1| 0| 0| 0| Ry REGISTER| 1| 0| 0| 0| 0| R/M| Rx REGISTER|
-----

```

Ry specifies destination register.

Rx specifies source register.

If R/M = 0: Rx and Ry are data registers.

If R/M = 1: Rx and Ry are address registers used for the pre-decrementing.

## RESULT

X - Set the same as the carry bit.

N - Undefined

Z - Cleared if the result is non-zero. Unchanged otherwise.

V - Undefined

C - Set if a decimal carry was generated. Cleared otherwise.

## SEE ALSO

SUB ADDI ADDQ  
 ADDX  
 ADD SUBI SUBQ  
 ABCD SUBX

## 1.126 Conditional Set

NAME

Scc -- Conditional set

SYNOPSIS

Scc <ea>

Size = (Byte)

FUNCTION

If condition is true then byte addressed by <ea> is set to \$FF, else byte addressed by <ea> is set to \$00.

Condition code 'cc' specifies one of the following:

0000	F	False	Z = 1	1000	VC	oVerflow Clear	V = 0
0001	T	True	Z = 0	1001	VS	oVerflow Set	V = 1
0010	HI	HIGH	C + Z = 0	1010	PL	PLus	N = 0
0011	LS	Low or Same	C + Z = 1	1011	MI	MINus	N = 1
0100	CC	Carry Clear	C = 0	1100	GE	Greater or Equal	N (+) V = 0
0101	CS	Carry Set	C = 1	1101	LT	Less Than	N (+) V = 1
0110	NE	Not Equal	Z = 0	1110	GT	Greater Than	Z + (N (+) V) = 0
0111	EQ	EQual	Z = 1	1111	LE	Less or Equal	Z + (N (+) V) = 1

FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
---	---	---	---	-----	---	---	-----	-----	-----	-----	-----	-----	-----	-----	-----
0	1	0	1	cc	CONDITION	1	1	MODE	REGISTER						

<ea>

REGISTER

<ea> specifies operand to set, addressing modes allowed are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

-----  
 RESULT  
 None.

SEE ALSO  
 Bcc DBcc

## 1.127 Stop processor execution (PRIVILEGED)

NAME  
 STOP -- Stop processor execution (PRIVILEGED)

SYNOPSIS  
 STOP #<data:16>

FUNCTION  
 Immediate data is moved to SR. PC is set to next instruction, and the processor stops fetch and execution of instruction. Execution restarts if if a TRACE exception, an interruption, or a RESET takes place.  
 When STOP is executed, a TRACE exception is generated (if T = 1). An interruption is allowed if it level is higher than current one. An external RESET always will generate a RESET exception. If bit S is set to zero by the immediate data, execution of this instruction will generate a "privilege violation".

FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
|-----|
|                                     16 BITS IMMEDIATE DATA                                     |
|-----
```

RESULT  
 SR is set according to immediate data.

SEE ALSO  
 RESET

## 1.128 SUBtract

NAME  
 SUB -- Subtract

SYNOPSIS  
 SUB <ea>,Dn  
 SUB Dn,<ea>

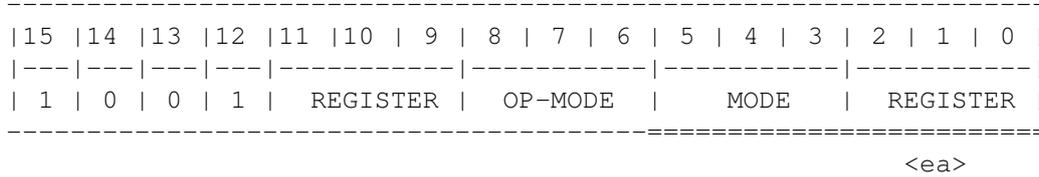
Size = (Byte, Word, Long)

---

FUNCTION

Subtracts source operand to destination operand.  
Result is stored to destination's place.

FORMAT



OP-MODE

Byte	Word	Long	
~~~~	~~~~	~~~~	
000	001	010	(Dn) - (<ea>) -> Dn
100	101	110	(<ea>) - (Dn) -> <ea>

REGISTER

One of the 8 datas registers

If <ea> is source, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An *	001	N\textdegree{} reg. An	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	111	010
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	111	011
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	111	011
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	111	011
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	111	011
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	111	100
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

\* Word or Long only

If <ea> is destination, allowed addressing modes are:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	-	-	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-

(An)+	011	N\textdegree	reg. An	(d8,PC,Xi)	-	-	
-(An)	100	N\textdegree	reg. An	(bd,PC,Xi)	-	-	
(d16,An)	101	N\textdegree	reg. An	([bd,PC,Xi],od)	-	-	
(d8,An,Xi)	110	N\textdegree	reg. An	([bd,PC],Xi,od)	-	-	
(bd,An,Xi)	110	N\textdegree	reg. An	#data	-	-	
([bd,An,Xi]od)	110	N\textdegree	reg. An				
([bd,An],Xi,od)	110	N\textdegree	reg. An				

When destination is an Address Register, SUBA instruction is used.

RESULT

- X - Set the same as the carry bit.
- N - Set if the result is negative. Cleared otherwise.
- Z - Set if the result is zero. Cleared otherwise.
- V - Set if an overflow is generated. Cleared otherwise.
- C - Set if a carry is generated. Cleared otherwise.

SEE ALSO

- ADDI ADDQ ADDX
- ADD SUBI SUBQ
- SUBX

### 1.129 SUBtract Address

NAME

SUBA -- Subtract address

SYNOPSIS

SUBA <ea>,An

Size = (Word, Long)

FUNCTION

Subtracts source operand to destination operand.  
 Source operand with a Word size is extended to 32 bits before operation. Result is stored to destination's place.

FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	REGISTER				OP-MODE		MODE		REGISTER			

<ea>

OP-MODE

- Indicates operation lenght:
- 011->one Word operation: source operand is extended to 32 bits
- 111->one Long operation



0	0	0	0	0	1	0	0	SIZE	MODE	REGISTER
16 BITS DATA (with last Byte)					8 BITS DATA					
32 BITS DATA (included last Word)										

## SIZE

00->one Byte operation  
 01->one Word operation  
 10->one Long operation

## REGISTER

<ea> is always destination, addressing modes are the followings:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	N\textdegree{} reg. Dn	Abs.W	111	000
An	-	-	Abs.L	111	001
(An)	010	N\textdegree{} reg. An	(d16,PC)	-	-
(An)+	011	N\textdegree{} reg. An	(d8,PC,Xi)	-	-
-(An)	100	N\textdegree{} reg. An	(bd,PC,Xi)	-	-
(d16,An)	101	N\textdegree{} reg. An	([bd,PC,Xi],od)	-	-
(d8,An,Xi)	110	N\textdegree{} reg. An	([bd,PC],Xi,od)	-	-
(bd,An,Xi)	110	N\textdegree{} reg. An	#data	-	-
([bd,An,Xi]od)	110	N\textdegree{} reg. An			
([bd,An],Xi,od)	110	N\textdegree{} reg. An			

## RESULT

X - Set the same as the carry bit.  
 N - Set if the result is negative. Cleared otherwise.  
 Z - Set if the result is zero. Cleared otherwise.  
 V - Set if an overflow is generated. Cleared otherwise.  
 C - Set if a carry is generated. Cleared otherwise.

## SEE ALSO

ADD ADDQ ADDX  
 SUB ADDI SUBQ  
 SUBX

## 1.131 SUBtract 3-bit immediate Quick

## NAME

SUBQ -- Subtract 3-bit immediate quick

## SYNOPSIS

```
SUBQ #<data>, <ea>
```

Size = (Byte, Word, Long)

## FUNCTION

Subtracts the immediate value of 1 to 8 to the operand at the destination location. The size of the operation may be specified as byte, word, or long. When subtracting to address registers, the condition codes are not altered, and the entire destination address register is used regardless of the operation size.

## FORMAT

```
-----
|15|14|13|12|11|10|9|8|7|6|5|4|3|2|1|0|
|---|---|---|---|-----|---|-----|-----|
| 0 | 1 | 0 | 1 |   DATA   | 1 | SIZE   |   MODE   | REGISTER |
=====
                                     <ea>
```

## DATA

000 ->represent value 8  
001 to 111 ->immediate data from 1 to 7

## SIZE

00->one Byte operation  
01->one Word operation  
10->one Long operation

## REGISTER

<ea> is always destination, addressing modes are the followings:

```
-----
|Addressing Mode|Mode| Register | |Addressing Mode|Mode|Register|
|-----|-----|-----| |-----|-----|-----|
|      Dn      |000| N\textdegree{} reg. Dn| |   Abs.W     |111| 000 |
|-----|-----|-----| |-----|-----|-----|
|      An *    |001| N\textdegree{} reg. An| |   Abs.L     |111| 001 |
|-----|-----|-----| |-----|-----|-----|
|      (An)    |010| N\textdegree{} reg. An| | (d16,PC)   | - | -   |
|-----|-----|-----| |-----|-----|-----|
|      (An)+   |011| N\textdegree{} reg. An| | (d8,PC,Xi) | - | -   |
|-----|-----|-----| |-----|-----|-----|
|      -(An)   |100| N\textdegree{} reg. An| | (bd,PC,Xi) | - | -   |
|-----|-----|-----| |-----|-----|-----|
|      (d16,An)|101| N\textdegree{} reg. An| | ([bd,PC,Xi],od)| - | -   |
|-----|-----|-----| |-----|-----|-----|
|      (d8,An,Xi)|110| N\textdegree{} reg. An| | ([bd,PC],Xi,od)| - | -   |
|-----|-----|-----| |-----|-----|-----|
|      (bd,An,Xi)|110| N\textdegree{} reg. An| | #data      | - | -   |
|-----|-----|-----| |-----|-----|-----|
| ([bd,An,Xi]od)|110| N\textdegree{} reg. An|
|-----|-----|-----|
| ([bd,An],Xi,od)|110| N\textdegree{} reg. An|
|-----|-----|-----|
```

\* Word or Long only.

## RESULT

X - Set the same as the carry bit.  
 N - Set if the result is negative. Cleared otherwise.  
 Z - Set if the result is zero. Cleared otherwise.  
 V - Set if an overflow is generated. Cleared otherwise.  
 C - Set if a carry is generated. Cleared otherwise.

## SEE ALSO

ADD ADDI  
 SUB SUBI ADDQ

## 1.132 SUBtract with eXtend

## NAME

SUBX -- Subtract with extend

## SYNOPSIS

SUBX Dy,Dx  
 SUBX -(Ay),-(Ax)

Size = (Byte, Word, Long)

## FUNCTION

Subtracts the source operand to the destination operand along with the extend bit, and stores the result in the destination location. The subtraction is performed using binary coded decimal arithmetic. The operands, which are packed BCD numbers, can be addressed in two different ways:

1. Data register to data register: The operands are contained in the data registers specified in the instruction.
2. Memory to memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

The size of operation can be specified as byte, word, or long.

Normally the Z condition code bit is set via programming before the start of an operation. That allows successful tests for zero results upon completion of multiple-precision operations.

## FORMAT

```
-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|-----|---|-----|---|---|-----|
| 1| 0| 0| 1|   Rx   | 1| SIZE | 0| 0|R/M|   Ry   |
-----
```

R/M = 0 -> data register  
 R/M = 1 -> address register  
 Rx: destination register  
 Ry: source register

## SIZE

00->one Byte operation  
 01->one Word operation  
 10->one Long operation

#### RESULT

X - Set the same as the carry bit.  
 N - Set if the result is negative. Cleared otherwise.  
 Z - Cleared if the result is non-zero. Unchanged otherwise.  
 V - Set if an overflow is generated. Cleared otherwise.  
 C - Set if a carry is generated. Cleared otherwise.

#### SEE ALSO

ADD ADDI  
 SUB SUBI ADDX

## 1.133 SWAP register upper and lower words

#### NAME

SWAP -- Swap register upper and lower words

#### SYNOPSIS

SWAP Dn

Size = (Word)

#### FUNCTION

Swaps between 16 low bits and 16 high bits of register.

#### FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | REGISTER |
-----
```

"REGISTER" indicates the number of register on which swap is made.

#### RESULT

X - Not affected  
 N - Set if the most-significant bit of the result was set. Cleared otherwise.  
 Z - Set if the 32 bits result was zero. Cleared otherwise.  
 V - Always cleared.  
 C - Always cleared.

#### SEE ALSO

EXG

## 1.134 Test And Set operand

#### NAME

TAS -- Test and set operand

## SYNOPSIS

TAS &lt;ea&gt;

Size = (Byte)

## FUNCTION

Test of a byte addressed by <ea>, bits N and Z of SR are set according to result of test.

Bit 7 of byte is set to 1. This instruction uses read-modify-write cycle, which is not dividable and allows synchronisation of several processors. But this instruction is NOT ALLOWED ON AMIGA!

This instruction can easily be substituted by BSET.

## FORMAT

```

-----
|15|14|13|12|11|10|9|8|7|6|5|4|3|2|1|0|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|0|1|0|0|1|0|1|0|1|1|MODE|REGISTER|
-----
                                     <ea>

```

## REGISTER

<ea> is destination, addressing modes are the followings:

```

-----
|Addressing Mode|Mode| Register | |Addressing Mode|Mode|Register|
|-----|-----|-----| |-----|-----|-----|
|      Dn      |000| N\textdegree{} reg. Dn| |  Abs.W      |111| 000 |
|-----|-----|-----| |-----|-----|-----|
|      An      | - | - | |  Abs.L      |111| 001 |
|-----|-----|-----| |-----|-----|-----|
|      (An)     |010| N\textdegree{} reg. An| | (d16,PC)   | - | - |
|-----|-----|-----| |-----|-----|-----|
|      (An)+    |011| N\textdegree{} reg. An| | (d8,PC,Xi) | - | - |
|-----|-----|-----| |-----|-----|-----|
|      -(An)    |100| N\textdegree{} reg. An| | (bd,PC,Xi) | - | - |
|-----|-----|-----| |-----|-----|-----|
|      (d16,An) |101| N\textdegree{} reg. An| | ([bd,PC,Xi],od) | - | - |
|-----|-----|-----| |-----|-----|-----|
|      (d8,An,Xi) |110| N\textdegree{} reg. An| | ([bd,PC],Xi,od) | - | - |
|-----|-----|-----| |-----|-----|-----|
|      (bd,An,Xi) |110| N\textdegree{} reg. An| | #data      | - | - |
|-----|-----|-----| |-----|-----|-----|
| ([bd,An,Xi]od) |110| N\textdegree{} reg. An|
|-----|-----|-----|
| ([bd,An],Xi,od) |110| N\textdegree{} reg. An|
-----

```

## RESULT

X - Not affected.

N - Set if MSB of byte is set. Cleared otherwise.

Z - Set if byte is zero. Cleared otherwise.

V - Always cleared.

C - Always cleared.

## SEE ALSO

## 1.135 Initiate processor TRAP

### NAME

TRAP -- Initiate processor trap

### SYNOPSIS

TRAP #<number>

### FUNCTION

Processor starts an exception process. TRAP number is pointed out by 4 bits into the instruction. 16 vectors are free to be used for TRAP (vectors from 32 to 47).

So the <number> can go from 0 to 15.

PC and SR are stored to SSP, and Vector is written to PC.

### FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | N\textdegree{} of TRAP |
-----
```

### RESULT

None.

### SEE ALSO

TRAPcc

## 1.136 Conditional trap

### NAME

TRAPcc -- Conditional trap (68020+)

### SYNOPSIS

TRAPcc  
 TRAPcc.w #<data>  
 TRAPcc.l #<data>

### FUNCTION

If "cc CONDITION" is true then there's generation of a level 7 exception, else execution continue normally.

Immediate data is optional, if given, the exception sub-routine can use it.

Condition code 'cc' specifies one of the following:

0000	F	False	Z = 1	1000	VC oVerflow Clear	V = 0
0001	T	True	Z = 0	1001	VS oVerflow Set	V = 1
0010	HI	High	C + Z = 0	1010	PL Plus	N = 0
0011	LS	Low or Same	C + Z = 1	1011	MI Minus	N = 1
0100	CC	Carry Clear	C = 0	1100	GE Greater or Equal	N (+) V = 0
0101	CS	Carry Set	C = 1	1101	LT Less Than	N (+) V = 1
0110	NE	Not Equal	Z = 0	1110	GT Greater Than	Z + (N (+) V) = 0
0111	EQ	Equal	Z = 1	1111	LE Less or Equal	Z + (N (+) V) = 1



## NAME

TST -- Test operand for zero

## SYNOPSIS

TST <ea>

Size = (Byte, Word, Long)

## FUNCTION

Operand is compared with zero. Flags are set according to the result.

## FORMAT

```

-----
|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|---|---|---|---|---|---|---|---|-----|-----|-----|
| 0| 1| 0| 0| 1| 0| 1| 0| SIZE | MODE | REGISTER |
-----=====
<ea>

```

## SIZE

00->one Byte operation

01->one Word operation

10->one Long operation

## REGISTER

<ea> is destination, if size is 16 or 32 bits then all addressing modes are allowed. If size is 8 bits, allowed addressing modes are:

```

-----
|Addressing Mode|Mode| Register | |Addressing Mode|Mode|Register|
|-----|-----|-----| |-----|-----|-----|
|      Dn      |000| N\textdegree{} reg. Dn| | Abs.W      |111| 000 |
|-----|-----|-----| |-----|-----|-----|
|      An      | - | - | | Abs.L      |111| 001 |
|-----|-----|-----| |-----|-----|-----|
|      (An)     |010| N\textdegree{} reg. An| | (d16,PC)   |111| 010 |
|-----|-----|-----| |-----|-----|-----|
|      (An)+    |011| N\textdegree{} reg. An| | (d8,PC,Xi) |111| 011 |
|-----|-----|-----| |-----|-----|-----|
|      -(An)    |100| N\textdegree{} reg. An| | (bd,PC,Xi) |111| 011 |
|-----|-----|-----| |-----|-----|-----|
|      (d16,An) |101| N\textdegree{} reg. An| | ([bd,PC,Xi],od)|111| 011 |
|-----|-----|-----| |-----|-----|-----|
|      (d8,An,Xi) |110| N\textdegree{} reg. An| | ([bd,PC],Xi,od)|111| 011 |
|-----|-----|-----| |-----|-----|-----|
|      (bd,An,Xi) |110| N\textdegree{} reg. An| | #data      | - | - |
|-----|-----|-----| |-----|-----|-----|
| ([bd,An,Xi]od) |110| N\textdegree{} reg. An|
|-----|-----|-----|
| ([bd,An],Xi,od) |110| N\textdegree{} reg. An|
-----

```

## RESULT

X - Not affected.

N - Set if the result is negative. Cleared otherwise.

Z - Set if the result is zero. Cleared otherwise.

V - Always cleared.

C - Always cleared.

SEE ALSO

BTST BFTST

## 1.139 Free stack frame created by LINK

NAME

UNLK -- Free stack frame created by LINK

SYNOPSIS

UNLK An

FUNCTION

This instruction does the inverse process of LINK instruction.  
Address register specified is moved in SP.  
Contents of SP is moved into address register.

FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | REGISTER |
-----
```

"REGISTER" indicates the number of address register, used as area pointer.

RESULT

None.

SEE ALSO

LINK

## 1.140 Unpack binary coded decimal

NAME

UNPK -- Unpack binary coded decimal (68020+)

SYNOPSIS

UNPK -(Ax), -(Ay), #<adjustment>

UNPK Dx, Dy, #<adjustment>

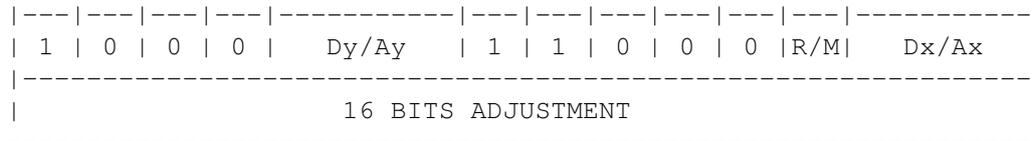
No size specs

FUNCTION

Convert packed two-digit-per-byte BCD to byte-per-digit unpacked BCD.

FORMAT

```
-----
|15 |14 |13 |12 |11 |10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
-----
```



R/M = 0 -> data register.

R/M = 1 -> Memory by pre-decrementing.

Register Dy/Ay specifies destination register.

Register Dx/Ax specifies source register.

"16 BITS ADJUSTMENT" is an immediate value added to source operand.

RESULT

None.

SEE ALSO

PACK

## 1.141 Move Instruction Execution Times

These following two tables indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as (r/w).

Move Byte and Word Instruction Execution Times

	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,ix)	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
An	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
(An)	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
(An)+	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
-(An)	10(2/0)	10(2/0)	14(2/1)	14(2/1)	14(2/1)	18(3/1)	20(4/1)	18(3/1)	22(4/1)
d(An)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
d(An,ix)	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
xxx.W	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
xxx.L	16(4/0)	16(4/0)	20(4/1)	20(4/1)	20(4/1)	24(5/1)	26(5/1)	24(5/1)	28(6/1)
d(PC)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
d(PC,ix)	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
#xxx	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)

The size of the index register (ix) does not affect execution time

Move Long Instruction Execute Times

	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,ix)	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
(An)	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
(An)+	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
-(An)	14(3/0)	14(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)

```

d(An) 16(4/0) 16(4/0) 24(4/2) 24(4/2) 24(4/2) 28(5/2) 30(5/2) 28(5/2) 32(6/2)
d(An,ix)18(4/0) 18(4/0) 26(4/2) 26(4/2) 26(4/2) 30(5/2) 32(5/2) 30(5/2) 34(6/2)
xxx.W 16(4/0) 16(4/0) 24(4/2) 24(4/2) 24(4/2) 28(5/2) 30(5/2) 28(5/2) 32(6/2)
xxx.L 20(5/0) 20(5/0) 28(5/2) 28(5/2) 28(5/2) 32(6/2) 34(6/2) 32(6/2) 36(7/2)
d(PC) 16(4/0) 16(4/0) 24(4/2) 24(4/2) 24(4/2) 28(5/2) 30(5/2) 28(5/2) 32(5/2)
d(PC,ix)18(4/0) 18(4/0) 26(4/2) 26(4/2) 26(4/2) 30(5/2) 32(5/2) 30(5/2) 34(6/2)
#xxx 12(3/0) 12(3/0) 20(3/2) 20(3/2) 20(3/2) 24(4/2) 26(4/2) 24(4/2) 28(5/2)

```

The size of the index register (ix) does not affect execution time

## 1.142 Standard Instruction Execution Times

The number of clock periods shown in this table indicates the time required to perform the operations, store the results and read the next instruction. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In the following table the headings have the following meanings:

An = address register operand, Dn = data register operand, ea = an operand specified by an effective address, and M = memory effective address operand.

### Standard Instruction Execution Times

instruction Size      op<ea>,An ^ op<ea>,Dn op Dn,<M>

```

ADD  byte,word 8(1/0) +      4(1/0) +      8(1/1) +
      long      6(1/0) +**      6(1/0) +**      12(1/2) +
AND  byte,word      -      4(1/0) +      8(1/1) +
      long      -      6(1/0) +**      12(1/2) +
CMP  byte,word 6(1/0) +      4(1/0) +      -
      long      6(1/0) +      6(1/0) +      -
DIVS      -      -      158(1/0) +*      -
DIVU      -      -      140(1/0) +*      -
EOR  byte,word      -      4(1/0) +**      8(1/1) +
      long      -      8(1/0) +**      12(1/2) +
MULS      -      -      70(1/0) +*      -
MULU      -      -      70(1/0) +*      -
OR   byte,word      -      4(1/0) +**      8(1/1) +
      long      -      6(1/0) +**      12(1/2) +
SUB  byte,word 8(1/0) +      4(1/0) +      8(1/1) +
      long      6(1/0) +**      6(1/0) +**      12(1/2) +

```

notes: + Add effective address calculation time

^ Word or long only

\* Indicates maximum value

\*\* The base time of six clock periods is increased to eight if the effective address mode is register direct or immediate (effective address time should also be added)

\*\*\* Only available effective address mode is data register direct

DIVS, DIVU - The divide algorithm used by the MC68000 provides less than 10% difference between the best and the worst case

timings.

MULS, MULU - The multiply algorithm requires  $38+2n$  clocks where  $n$  is defined as:

MULU:  $n$  = the number of ones in the <ea>

MULS:  $n$  = concatenate the <ea> with a zero as the LSB;

$n$  is the resultant number of 10 or 01 patterns in the 17-bit source; i.e., worst case happens when the source is \$5555

### 1.143 Immediate Instruction Execution Times

The number of clock periods shown in this table includes the time to fetch immediate operands, perform the operations, store the results and read the next operation. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Immediate Instruction Execution Times

instruction	size	op #, Dn	op #, An	op #, M
ADDI	byte, word	8 (2/0)	-	12 (2/1) +
	long	16 (3/0)	-	20 (3/2) +
ADDQ	byte, word	4 (1/0)	8 (1/0) *	8 (1/1) +
	long	8 (1/0)	8 (1/0)	12 (1/2) +
ANDI	byte, word	8 (2/0)	-	12 (2/1) +
	long	16 (3/0)	-	20 (3/1) +
CMPI	byte, word	8 (2/0)	-	8 (2/0) +
	long	14 (3/0)	-	12 (3/0) +
EORI	byte, word	8 (2/0)	-	12 (2/1) +
	long	16 (3/0)	-	20 (3/2) +
MOVEQ	long	4 (1/0)	-	-
ORI	byte, word	8 (2/0)	-	12 (2/1) +
	long	16 (3/0)	-	20 (3/2) +
SUBI	byte, word	8 (2/0)	-	12 (2/1) +
	long	16 (3/0)	-	20 (3/2) +
SUBQ	byte, word	4 (1/0)	8 (1/0) *	8 (1/1) +
	long	8 (1/0)	8 (1/0)	12 (1/2) +

+ Add effective address calculation time

\* word only

### 1.144 Single Operand Instruction Execution Times

This table indicates the number of clock periods for the single operand instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

## Single Operand Instruction Execution Times

	instruction size	register	memory
CLR	byte, word 4 (1/0)	8 (1/1) +	
	long 6 (1/0)	12 (1/2) +	
NBCD	byte 6 (1/0)	8 (1/1) +	
NEG	byte, word 4 (1/0)	8 (1/1) +	
	long 6 (1/0)	12 (1/2) +	
NEGX	byte, word 4 (1/0)	8 (1/1) +	
	long 6 (1/0)	12 (1/2) +	
NOT	byte, word 4 (1/0)	8 (1/1) +	
	long 6 (1/0)	12 (1/2) +	
Scc	byte, false 4 (1/0)	8 (1/1) +	
	byte, true 6 (1/0)	8 (1/1) +	
TAS #	byte 4 (1/0)	10 (1/1) +	
TST	byte, word 4 (1/0)	4 (1/0) +	
	long 4 (1/0)	4 (1/0) +	

+ add effective address calculation time

# This instruction should never be used on the Amiga as its invisible read/write cycle can disrupt system DMA.

**1.145 Rotate Instruction Execution Times**

This table indicates the number of clock periods for the shift and rotate instructions. The number of read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

## Shift/Rotate Instruction Execution Times

	instruction size	register	memory
ASR, ASL	byte, word 6+2n (1/0)	8 (1/1) +	
	long 8+2n (1/0)	-	
LSR, LSL	byte, word 6+2n (1/0)	8 (1/1) +	
	long 8+2n (1/0)	-	
ROR, ROL	byte, word 6+2n (1/0)	8 (1/1) +	
	long 8+2n (1/0)	-	
ROXR, ROXL	byte, word 6+2n (1/0)	8 (1/1) +	
	long 8+2n (1/0)	-	

+ add effective address calculation time

n is the shift or rotate count

**1.146 Bit Manipulation Instruction Execution Times**

This table indicates the number of clock periods required for the bit manipulation instructions. The number of read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

#### Bit Manipulation Instruction Execution Times

instruction	size	dynamic		static	
		register	memory	register	memory
BCHG	byte	-	8(1/1) +	-	12(2/1) +
	long	8(1/0) *	-	12(2/0) *	-
BCLR	byte	-	8(1/1) +	-	12(2/1) +
	long	10(1/0) *	-	14(2/0) *	-
BSET	byte	-	8(1/1) +	-	12(2/1) +
	long	8(1/0) *	-	12(2/0) *	-
BTST	byte	-	4(1/0) +	-	8(2/0) +
	long	6(1/0)	-	10(2/0)	-

+ add effective address calculation time

\* indicates maximum value

## 1.147 Specificational Instruction Execution Times

This table indicates the number of clock periods for the conditional instructions. The number of read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

#### Conditional Instruction Execution Times

instruction	displacement	branch	
		taken	not taken
Bcc	byte	10(2/0)	8(1/0)
	word	10(2/0)	12(1/0)
BRA	byte	10(2/0)	-
	word	10(2/0)	-
BSR	byte	18(2/2)	-
	word	18(2/2)	-
DBcc	CC true	-	12(2/0)
	CC false	10(2/0)	14(3/0)

## 1.148 JMP, JSR, LEA, PEA and MOVEM Instruction Execution Times

This Table indicates the number of clock periods required for the jump, jump-to-subroutine, load effective address, push effective address and move multiple registers instructions. The number of bus read and write

cycles is shown in parenthesis as (r/w).

#### JMP, JSR, LEA, PEA and MOVEM Instruction Execution Times

instr	size	(An)	(An)+	-(An)	d(An)	
JMP	-	8 (2/0)	-	-	10 (2/0)	
JSR	-	16 (2/2)	-	-	18 (2/2)	
LEA	-	4 (1/0)	-	-	8 (2/0)	
PEA	-	12 (1/2)	-	-	16 (2/2)	
MOVEM	word	12+4n	12+4n	-	16+4n	
M->R		(3+n/0)	(3+n/0)	-	(4+n/0)	
	long	12+8n	12+8n	-	16+8n	
		(3+2n/0)	(3+2n/0)	-	(4+2n/0)	
MOVEM	word	8+4n	-	8+4n	12+4n	
R->M		(2/n)	-	(2/n)	(3/n)	
	long	8+8n	-	8+8n	12+8n	
		(2/2n)	-	(2/2n)	(3/2n)	
instr	size	d(An,ix)+	xxx.W	xxx.L	d(PC)	d(PC,ix)*
JMP	-	14 (3/0)	10 (2/0)	12 (3/0)	10 (2/0)	14 (3/0)
JSR	-	22 (2/2)	18 (2/2)	20 (3/2)	18 (2/2)	22 (2/2)
LEA	-	12 (2/0)	8 (2/0)	12 (3/0)	8 (2/0)	12 (2/0)
PEA	-	20 (2/2)	16 (2/2)	20 (3/2)	16 (2/2)	20 (2/2)
MOVEM	word	18+4n	16+4n	20+4n	16+4n	18+4n
M->R		(4+n/0)	(4+n/0)	(5+n/0)	(4+n/0)	(4+n/0)
	long	18+8n	16+8n	20+8n	16+8n	18+8n
		(4+2n/0)	(4+2n/0)	(5+2n/0)	(4+2n/0)	(4+2n/0)
MOVEM	word	14+4n	12+4n	16+4n	-	-
R->M		(3/n)	(3/n)	(4/n)	-	-
	long	14+8n	12+8n	16+8n	-	-
		(3/2n)	(3/2n)	(4/2n)	-	-

n is the number of registers to move

\* is the size of the index register (ix) does not affect the instruction's execution time

## 1.149 Multi-Precision Instruction Execution Times

This table indicates the number of clock periods for the multi-precision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results and read the next instructions. The number of read and write cycles is shown in parenthesis as (r/w).

The headings have the following meanings: Dn = data register operand and M = memory operand.

#### Multi-Precision Instruction Execution Times

instruction	size	op Dn,Dn	op M,M
ADDX	byte,word	4 (1/0)	18 (3/1)
	long	8 (1/0)	30 (5/2)

CMPM	byte, word	-	12 (3/0)
	long	-	20 (5/0)
SUBX	byte, word	4 (1/0)	18 (3/1)
	long	8 (1/0)	30 (5/2)
ABCD	byte	6 (1/0)	18 (3/1)
SBCD	byte	6 (1/0)	18 (3/1)

## 1.150 Miscellaneous Instruction Execution Times

This table indicates the number of clock periods for the following miscellaneous instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

### Miscellaneous Instruction Execution Times

instruction	size	register	memory
ANDI to CCR	byte	20 (3/0)	-
ANDI to SR	word	20 (3/0)	-
CHK	-	10 (1/0) +	-
EORI to CCR	byte	20 (3/0)	-
EORI to SR	word	20 (3/0)	-
ORI to CCR	byte	20 (3/0)	-
ORI to SR	word	20 (3/0)	-
MOVE from SR	-	6 (1/0)	8 (1/1) +
MOVE to CCR	-	12 (1/0)	12 (1/0) +
MOVE to SR	-	12 (1/0)	12 (1/0) +
EXG	-	6 (1/0)	-
EXT	word	4 (1/0)	-
	long	4 (1/0)	-
LINK	-	16 (2/2)	-
MOVE from USP	-	4 (1/0)	-
MOVE to USP	-	4 (1/0)	-
NOP	-	4 (1/0)	-
RESET	-	132 (1/0)	-
RTE	-	20 (5/0)	-
RTR	-	20 (5/0)	-
RTS	-	16 (4/0)	-
STOP	-	4 (0/0)	-
SWAP	-	4 (1/0)	-
TRAPV (No Trap)	-	4 (1/0)	-
UNLK	-	12 (3/0)	-

+ add effective address calculation time

## 1.151 Move Peripheral Instruction Execution Times

instruction size register->memory memory->register

```

MOVEP   word  16(2/2)    16(4/0)
        long  24(2/4)    24(6/0)

```

## 1.152 Exception Processing Execution Times

This table indicates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch and the fetch of the first two instruction words of the handler routine. The number of bus read and write cycles is shown in parenthesis as (r/w).

### Exception Processing Execution Times

exception	periods
address error	50(4/7)
bus error	50(4/7)
CHK instruction (trap taken)	44(5/3)+
Divide by Zero	42(5/3)
illegal instruction	34(4/3)
interrupt	44(5/3)*
privilege violation	34(4/3)
RESET **	40(6/0)
trace	34(4/3)
TRAP instruction	38(4/3)
TRAPV instruction (trap taken)	34(4/3)

+ add effective address calculation time

\* the interrupt acknowledge cycle is assumed to take four clock periods

\*\* indicates the time from when RESET and HALT are first sampled as negated to when instruction execution starts

## 1.153 ASP68K PROJECT, Sixth Edition

ASP68K PROJECT

Sixth Edition

by Michael Glew  
 mglew@laurel.ocs.mq.edu.au  
 Technophilia BBS +61-2-8073563

January 1994

-----  
 C O N T R I B U T O R S  
 -----

Erik Bakke, Robert Barton, Bernd Blank, Kasimir Blomstedt, Frans Bouma, David Carson, Nicolas Dade, Aaron Digulla, Irmen de Jong, Andy Duplain, Denis Duplan, Steven Eker, Calle Englund, Alexander Fritsch, Charlie Gibbs, Kurt Haenen, Jon Hudson, Kjetil Jacobsen, Olav Kalgraf, Makoto Kamada, Markku Kolkka, John Lane, Jonathan Mahaffy, Dave Mc Mahan, Lindsay Meek, Walter Misar, Boerge Noest, Gunnar Rxnning, Jay Scott, Olaf Seibert, Peter Simons.

---

## I N T R O D U C T I O N

---

A while back, I was quite interested to find that there was an electronic magazine called "howtocode" that included lots of interesting hints and tips of coding. In the fifth edition, there was a list of optimizations that really got me thinking. "What if there was a proggy that you could put an assembler program through, that would speed it up, taking out all the stupid things output by compilers, and over-tired coders?" 8). I started combing the networks, and came across one such program, called the "SELCO Source Optimizer". It only had four optimizations, so I set to writing my own.

Step one was to collect as many optimization ideas as I could. I posted to Usenet and got an impressive response, and the contributors are listed above. I promised a report on the optimizations received, and here it is. My aim now is to write a program to make these optimizations, and to distribute it. Contributors will receive a copy of the final archive, to thank them for their time and energy. Further contributions will be welcomed, so rather than making changes yourself tell me what you want changed, and I'll distribute it with the next update.

---

## C H A N G E S

---

### 2nd Edition

The second edition incorporated a hell of a lot of corrections. Double copies of some optimizations were incorporated in to just one copy, and a few additions were made. Sorry that the first edition was not sent out to all contributors, but I was a tad busy. 8)

### 3rd Edition

Due to the distribution of the second edition document, many comments were received and a couple of the "optimizations" were found to be incorrect. Analysis of the mul/div optimizations ended in a few modifications for safety. They still save a huge number of clock cycles, so it is better to be safe than sorry.

Also, I have made it so that the number of words of space saved or

---

increased is shown. Space savings are positive, increases are negative. Zero means no change.

#### 4th Edition

Some minor changes and additions as well as the addition of columns for '030 and '040 CPUs - whole new format was required...

#### 5th Edition

Eric Bakke released his docs on 020+ CPUs and 881/882 FPU's. I have been given permission to use these docs to further the capabilities of asp68k. Thanks Eric... I really would like to get a hold of the 020,030,040 Programmer Reference Cards or manuals, so if anyone has any copies they wanna send me, let me know... Local Motorola Distributers are not too helpful.

#### 6th Edition

Aaron Digulla advised that it would be helpful if the optimizations were sorted somehow. I will sort by the the first letters of the first line of the optimizations. Also a special thanks to Makoto Kamada for his detailed contributions, without such this text would have died long ago..

---

## O P T I M I Z A T I O N S

---

#### Note:-

m? = memory operand  
 dx = data register  
 ds = data register (scratch)  
 ax = address register  
 rx = either a data or address register  
 #n = immediate operand  
 ??,?1,?2= address label  
 \* = anything  
 .x = any size  
 b<cc> = branch commands

Opt = optimization  
 Notes = notes about where optimization is valid, and misc notes  
 Speed = are clock periods saved? ("Y" = yes  
       "y" = in some cases  
       "N" = no  
       "\*" = increase  
        "-" = cannot be used on this cpu  
        "!" = must be used on this cpu)  
 Size = how many bytes are saved?

---

Opt	Speed			Size		
	000	010	020	030	040	
* ??* -> * n(pc)*		Y   Y   ?   ?   ?   2				
n = ??-pc, n < 32768						
*0(ax)* -> *(ax)*		Y   Y   ?   ?   ?   2				
add*.x #0,dx -> tst.x dx		Y   Y   ?   ?   ?   2/4				
add.x #n,* -> addq.x #n,*		Y   Y   ?   ?   ?   2/4				
if 1 <= n <= 8						
add.x #n,* -> subq.x #-n,*		Y   Y   ?   ?   ?   2/4				
-8 <= n <= -1						
add.x #n,ax -> lea n(ax),ax		Y   Y   ?   ?   ?   0/2				
-32767 <= n <= -9, 9 <= n <= 32767						
addq.l #n,ax -> addq.w #n,ax		Y   Y   ?   ?   ?   0				
addq.l #n,ry -> add.l #(n+m),ry		Y   Y   ?   ?   ?   -2				
addq.l #m,ry						
addq.x #2,ax -> move.w *,(ax)		Y   Y   ?   Y   ?   2				
move.w *,-(ax)						
.x is .w or .l						
addq.x #4,ax -> move.l *,(ax)		Y   Y   ?   Y   ?   2				
move.l *,-(ax)						
.x is .w or .l						
addq.x #6,ax -> move.w *1,4(ax)		Y   Y   ?   ?   ?   0				
move.w *1,-(ax)    move.l *2,(ax)						
move.l *2,-(ax)						
.x is .w or .l						
*1 and *2 do not contain ax						
addq.x #6,ax -> move.l *1,2(ax)		Y   Y   ?   ?   ?   0				
move.l *1,-(ax)    move.w *2,(ax)						
move.w *2,-(ax)						
.x is .w or .l						
*1 and *2 do not contain ax						
addq.x #8,ax -> move.l *1,4(ax)		Y   Y   ?   ?   ?   0				
move.l *1,-(ax)    move.l *2,(ax)						
move.l *2,-(ax)						
.x is .w or .l						

\*1 and \*2 do not contain ax

addq.x #4, sp -> move.l ax, (sp)	Y   Y   ?   Y   ?   2
pea (ax)	

.x is .w or .l

ax, ay are not a7(=sp)

addq.x #6, sp -> move.w *, 4(sp)	Y   Y   ?   ?   ?   0
move.w *, -(sp)      move.l ax, (sp)	
pea (ax)	

.x is .w or .l

ax, ay are not a7(=sp)

addq.x #6, sp -> move.l ax, 2(sp)	Y   Y   ?   ?   ?   0
pea (ax)              move.w *, (sp)	
move.w *, -(sp)	

.x is .w or .l

ax, ay are not a7(=sp)

addq.x #8, sp -> move.l *, 4(sp)	Y   Y   ?   ?   ?   0
move.l *, -(sp)      move.l ax, (sp)	
pea (ax)	

.x is .w or .l

ax, ay are not a7(=sp)

addq.x #8, sp -> move.l ax, 4(sp)	Y   Y   ?   ?   ?   0
pea (ax)              move.l *, (sp)	
move.l *, -(sp)	

.x is .w or .l

ax, ay are not a7(=sp)

addq.x #8, sp -> move.l ax, 4(sp)	Y   Y   ?   ?   ?   0
pea (ax)              move.l ay, (sp)	
pea (ay)	

.x is .w or .l

ax, ay are not a7(=sp)

and.l #n, dx -> bclr.l #b, dx	Y   Y   ?   ?   ?   2
-------------------------------	-----------------------

not(n) = 2^b (only 1 bit off)

asl.b #2, dy -> add.b dy, dy	Y   Y   ?   ?   ?   -2
add.b dy, dy	

asl.b #n, dx -> clr.b dx	Y   Y   ?   ?   ?   0
--------------------------	-----------------------

status flags are wrong, n>=8

asl.l #16, dx -> swap dx	Y   Y   ?   ?   ?   -2
clr.w dx	

```

status flags are wrong
-----+-----+-----+-----+-----+-----+-----+-----+
asl.l #n,dx -> asl.w #(n-16),dx      | Y | Y | ? | ? | ? | -4
      swap dx          | | | | |
      clr.w dx         | | | | |
-----+-----+-----+-----+-----+
status flags are wrong, 16<n<32
-----+-----+-----+-----+-----+
asl.l #n,dx -> moveq #0,dx          | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+
status flags are wrong, n>=32
-----+-----+-----+-----+-----+
asl.w #2,dy -> add.w dy,dy          | Y | Y | ? | ? | ? | -2
      add.w dy,dy      | | | | |
-----+-----+-----+-----+-----+
asl.w #n,dx -> clr.w dx              | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+
status flags are wrong, n>=16
-----+-----+-----+-----+-----+
asl.x #1,dy -> add.x dy,dy          | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+
asr.b #n,dx -> clr.b dx              | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+
status flags are wrong, n>=8
-----+-----+-----+-----+-----+
asr.l #16,dx -> swap dx              | Y | Y | ? | ? | ? | -2
      ext.l dx         | | | | |
-----+-----+-----+-----+-----+
status flags are wrong
-----+-----+-----+-----+-----+
asr.l #n,dx -> moveq #0,dx          | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+
status flags are wrong, n>=32
-----+-----+-----+-----+-----+
asr.l #n,dx -> swap dx              | Y | Y | ? | ? | ? | -4
      asr.w #(n-16),dx | | | | |
      ext.l dx         | | | | |
-----+-----+-----+-----+-----+
status flags are wrong, 16<n<32
-----+-----+-----+-----+-----+
asr.w #n,dx -> clr.w dx              | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+
status flags are wrong, n>=16
-----+-----+-----+-----+-----+
b<cc>.w ?? -> b<cc>.s ??            | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+
abs(??-pc)<128
-----+-----+-----+-----+-----+
bclr.l #n,dx -> and.w #m,dx         | Y | Y | ? | Y | ? | 0
-----+-----+-----+-----+-----+
0 <= n <= 15, m = 65535-(2^n)
status flags are wrong
-----+-----+-----+-----+-----+
bra ?? -> (nothing)                 | Y | Y | Y | ? | ? | 2/4
??   ??                             | | | | |
-----+-----+-----+-----+-----+
remove null branches, but keep the label

```

```

-----+-----+-----+-----+-----+-----+-----+
bset.b #7,m? -> tas m?          | y | y | ? | ? | ? | 2
beq ??          bpl ??          |   |   |   |   |   |   |
-----+-----+-----+-----+-----+
m? must be address allowing read-modify-write transfer.
Status flags are wrong
-----+-----+-----+-----+-----+
bset.b #7,m? -> tas m?          | y | y | ? | ? | ? | 2
bne ??          bmi ??          |   |   |   |   |   |   |
-----+-----+-----+-----+-----+
m? must be address allowing read-modify-write transfer.
Status flags are wrong
-----+-----+-----+-----+-----+
bset.b #7,m? -> tas m?          | y | y | ? | ? | ? | 2
-----+-----+-----+-----+-----+
m? must be address allowing read-modify-write transfer.
Status flags are wrong
-----+-----+-----+-----+-----+
bset.l #7,dx -> tas dx          | Y | Y | ? | Y | ? | 2
beq ??          bpl ??          |   |   |   |   |   |   |
-----+-----+-----+-----+-----+
status flags are wrong
-----+-----+-----+-----+-----+
bset.l #7,dx -> tas dx          | Y | Y | ? | Y | ? | 2
bne ??          bmi ??          |   |   |   |   |   |   |
-----+-----+-----+-----+-----+
status flags are wrong
-----+-----+-----+-----+-----+
bset.l #7,dx -> tas dx          | Y | Y | ? | Y | ? | 2
-----+-----+-----+-----+-----+
status flags are wrong
-----+-----+-----+-----+-----+
bset.l #n,dx -> or.w #m,dx      | Y | Y | ? | Y | ? | 0
-----+-----+-----+-----+-----+
0 <= n <= 15, m = 2^n
status flags are wrong
-----+-----+-----+-----+-----+
bsr ?? -> bra ??          | Y | Y | ? | ? | ? | 2
rts          | | | | |
-----+-----+-----+-----+-----+
different stack depth
-----+-----+-----+-----+-----+
btst.b #7,m? -> tst.b m?       | Y | Y | ? | ? | ? | 2
beq ??          bpl ??          |   |   |   |   |   |   |
-----+-----+-----+-----+-----+
Status flags are wrong. Not valid for Dn, d16(PC), d8(PC,Xn)
dest address modes.
-----+-----+-----+-----+-----+
btst.b #7,m? -> tst.b m?       | Y | Y | ? | ? | ? | 2
bne ??          bmi ??          |   |   |   |   |   |   |
-----+-----+-----+-----+-----+
Status flags are wrong. Not valid for Dn, d16(PC), d8(PC,Xn)
dest address modes.
-----+-----+-----+-----+-----+
btst.l #7,dx -> tst.b dx       | Y | Y | ? | Y | ? | 2
beq ??          bpl ??          |   |   |   |   |   |   |
-----+-----+-----+-----+-----+

```

Status flags are wrong.

btst.l #7,dx -> tst.b dx		Y		Y		?		Y		?		2
bne ??												

Status flags are wrong.

btst.l #15,dx -> tst.w dx		Y		Y		?		Y		?		2
beq ??												

Status flags are wrong.

btst.l #15,dx -> tst.w dx		Y		Y		?		Y		?		2
bne ??												

Status flags are wrong.

btst.l #31,dx -> tst.l dx		Y		Y		?		Y		?		2
beq ??												

Status flags are wrong.

btst.l #31,dx -> tst.l dx		Y		Y		?		Y		?		2
bne ??												

status flags are wrong

clr.b mn -> clr.w mn		Y		Y		?		?		?		2/4/6
clr.b mn+1												

best if mn is longword aligned

clr.l dx -> moveq #0,dx		Y		Y		?		?		?		0
-------------------------	--	---	--	---	--	---	--	---	--	---	--	---

clr.w mn -> clr.l mn		Y		Y		?		?		?		2/4/6
clr.w mn+2												

best if mn is longword aligned

clr.x -(ax) -> move.x ds,-(ax)		Y		Y		?		?		?		0
--------------------------------	--	---	--	---	--	---	--	---	--	---	--	---

ds must equal zero

clr.x n(ax,rx) -> move.x ds,n(ax,rx)		Y		Y		?		?		?		0
--------------------------------------	--	---	--	---	--	---	--	---	--	---	--	---

ds must equal zero

cmp.x #0,ax -> move.x ax,ds		Y		Y		?		?		?		2/4
-----------------------------	--	---	--	---	--	---	--	---	--	---	--	-----

move ax to scratch register

cmp.x #0,ax -> tst.x ax		-		-		?		?		?		?
-------------------------	--	---	--	---	--	---	--	---	--	---	--	---

for .w and .l

cmp.x #0,dx -> tst.x dx		Y		Y		?		?		?		2/4
-------------------------	--	---	--	---	--	---	--	---	--	---	--	-----

```
cmp.x #0,m? -> tst.x m?      | Y | Y | ? | ? | ? | 2/4
```

```
-----+-----+-----+-----+-----+-----
may not be legal on some early '000 CPUs
```

```
divu.l #n,dx -> lsr.l #m,dx  | ! | ! | ? | ? | ? | 4
```

```
-----+-----+-----+-----+-----+-----
n is 2^m, 1 <= m <= 8
```

```
divu.l #n,dx -> moveq #0,dx  | ! | ! | ? | ? | ? | 4
```

```
-----+-----+-----+-----+-----+-----
n is 2^m, m>=32
```

```
divu.l #n,dx -> moveq #m,ds  | ! | ! | ? | ? | ? | 2
lsr.l ds,dx   | | | | |
```

```
-----+-----+-----+-----+-----+-----
n is 2^m, 8<m<32
```

```
divu.w #n,dx -> lsr.l #m,dx  | Y | Y | ? | ? | ? | 2
```

```
-----+-----+-----+-----+-----+-----
n is 2^m, 1 <= m <= 8, ignore remainder
```

```
divu.w #n,dx -> moveq #0,dx  | Y | Y | ? | ? | ? | 2
```

```
-----+-----+-----+-----+-----+-----
n is 2^m, m>=32
```

```
divu.w #n,dx -> moveq #m,ds  | Y | Y | ? | ? | ? | 0
lsr.l ds,dx   | | | | |
```

```
-----+-----+-----+-----+-----+-----
n is 2^m, 8<m<32, ignore remainder
```

```
eor.x #-1,* -> not.x *      | Y | Y | ? | ? | ? | 2/4
```

```
-----+-----+-----+-----+-----+-----
ext.w dx -> extb.l dx       | - | - | ? | ? | ? | 2
ext.l dx   | | | | |
```

```
-----+-----+-----+-----+-----+-----
jmp ?? -> bra.w ??         | Y | Y | ? | ? | ? | 2
```

```
-----+-----+-----+-----+-----+-----
abs(??-pc) < 32768, same section
```

```
jsr * -> jmp *             | Y | Y | ? | ? | ? | 2
rts   | | | | |
```

```
-----+-----+-----+-----+-----+-----
different stack depth
```

```
jsr ?1 -> pea ?2          | y | y | ? | ? | ? | 0
jmp ?2   jmp ?1          | | | | |
```

```
-----+-----+-----+-----+-----+-----
same time if jsr is abs.l
```

```
-----+-----+-----+-----+-----+-----
jsr ?? -> bsr.w ??         | Y | Y | ? | ? | ? | 2
```

```
-----+-----+-----+-----+-----+-----
abs(??-pc) < 32768, same section
```

```
-----+-----+-----+-----+-----+-----
lea (ax),ax -> (nothing)   | Y | Y | Y | ? | ? | 2
```

```
-----+-----+-----+-----+-----+-----
delete
```

```

lea 0.w,ax -> sub.l ax,ax      | Y | Y | - | - | - | 2
-----+-----+-----+-----+-----+-----
lea n(ax),ax -> addq.w #n,ax   | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
if 1 <= n <= 8
-----+-----+-----+-----+-----+-----
lea n(ax),ax -> subq.w #-n,ax | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
if -8 <= n <= -1
-----+-----+-----+-----+-----+-----
lsl.b #2,dx -> add.b dy,dy     | Y | Y | ? | ? | ? | -2
      add.b dy,dy           | | | | |
-----+-----+-----+-----+-----+-----
lsl.b #n,dx -> clr.b dx       | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+-----
status flags are wrong, n>=8
-----+-----+-----+-----+-----+-----
lsl.l #16,dx -> swap dx       | Y | Y | ? | ? | ? | -2
      clr.w dx              | | | | |
-----+-----+-----+-----+-----+-----
status flags are wrong
-----+-----+-----+-----+-----+-----
lsl.l #n,dx -> lsl.w #(n-16),dx | Y | Y | ? | ? | ? | -4
      swap dx              | | | | |
      clr.w dx             | | | | |
-----+-----+-----+-----+-----+-----
status flags are wrong, 16<n<32
-----+-----+-----+-----+-----+-----
lsl.l #n,dx -> moveq #0,dx    | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+-----
status flags are wrong, n>=32
-----+-----+-----+-----+-----+-----
lsl.w #2,dx -> add.w dy,dy     | Y | Y | ? | ? | ? | -2
      add.w dy,dy         | | | | |
-----+-----+-----+-----+-----+-----
lsl.w #n,dx -> clr.w dx       | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+-----
status flags are wrong, n>=16
-----+-----+-----+-----+-----+-----
lsl.x #1,dx -> add.x dy,dy     | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+-----
lsr.b #n,dx -> clr.b dx       | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+-----
status flags are wrong, n>=8
-----+-----+-----+-----+-----+-----
lsr.l #16,dx -> clr.w dx       | Y | Y | ? | ? | ? | -2
      swap dx              | | | | |
-----+-----+-----+-----+-----+-----
status flags are wrong
-----+-----+-----+-----+-----+-----
lsr.l #n,dx -> clr.w dx       | Y | Y | ? | ? | ? | -4
      swap dx              | | | | |
      lsr.w #(n-16),dx     | | | | |
-----+-----+-----+-----+-----+-----
status flags are wrong, 16<n<32
-----+-----+-----+-----+-----+-----
lsr.l #n,dx -> moveq #0,dx    | Y | Y | ? | ? | ? | 0

```

```

-----+-----+-----+-----+-----+-----
status flags are wrong, n>=32
-----+-----+-----+-----+-----+-----
lsl.w #n,dx -> clr.w dx      | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+-----
status flags are wrong, n>=16
-----+-----+-----+-----+-----+-----
move.b #-1,(ax) -> st (ax)   | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
status flags are wrong
-----+-----+-----+-----+-----+-----
move.b #-1,(ax)+ -> st (ax)+ | N | N | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
status flags are wrong
-----+-----+-----+-----+-----+-----
move.b #-1,-(ax) -> st -(ax) | N | N | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
status flags are wrong
-----+-----+-----+-----+-----+-----
move.b #-1,?? -> st ??      | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
status flags are wrong
-----+-----+-----+-----+-----+-----
move.b #-1,dx -> st dx      | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
status flags are wrong
-----+-----+-----+-----+-----+-----
move.b #-1,n(ax) -> st n(ax) | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
status flags are wrong
-----+-----+-----+-----+-----+-----
move.b #-1,n(ax,rx) -> st n(ax,rx) | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
status flags are wrong
-----+-----+-----+-----+-----+-----
move.b #x,mn -> move.w #xy,mn | Y | Y | ? | ? | ? | 4/6/8
move.b #y,mn+1      | | | | |
-----+-----+-----+-----+-----+-----
best if mn is longword aligned
-----+-----+-----+-----+-----+-----
move.l #n,-(sp) -> pea n.w    | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
-32767 <= n <= 32767
-----+-----+-----+-----+-----+-----
move.l #n,ax -> move.w #n,ax  | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----
-32767 <= n <= 32767
-----+-----+-----+-----+-----+-----
move.l #n,dx -> moveq #-128,dx | Y | Y | ? | N | * | 2
subq.l #n+128,dx | | | | |
-----+-----+-----+-----+-----+-----
-136 <= n <= -129
-----+-----+-----+-----+-----+-----
move.l #n,dx -> moveq #m,dx   | Y | Y | ? | ? | ? | 2
not.b dx      | | | | |
-----+-----+-----+-----+-----+-----
128 <= n <= 255, m = 255-n

```

```

-----+-----+-----+-----+-----+-----+
move.l #n,dx -> moveq #m,dx      | Y | Y | ? | ? | ? | 2
      not.w dx      | |  | |  | |
                    | |  | |  | |
-----+-----+-----+-----+-----+
65534 <= n <= 65408 or -65409 <= n <= -65536, m = 65535-abs(n)
-----+-----+-----+-----+-----+
move.l #n,dx -> moveq #m,dx      | Y | Y | ? | ? | ? | 2
      swap dx      | |  | |  | |
                    | |  | |  | |
-----+-----+-----+-----+-----+
-8323073 <= n <= -65537 or 4096 <= n <= 8323072, n = m*65536
-----+-----+-----+-----+-----+
move.l #n,dx -> moveq #n,dx      | Y | Y | ? | ? | ? | 4
-----+-----+-----+-----+-----+
if -128 <= n <= 127
-----+-----+-----+-----+-----+
move.l #n,dx -> moveq #y,dx      | * | * | ? | ? | ? | 2
      lsl.l #z,dx  | |  | |  | |
-----+-----+-----+-----+-----+
n = y * 2^z
-----+-----+-----+-----+-----+
move.l #n,dx -> moveq #m,dx      | Y | Y | ? | N | ? | 2
      add.b dx,dx  | |  | |  | |
-----+-----+-----+-----+-----+
(128 <= n <= 254 or -256 <= n <= -130) and n is even, m = n/2
-----+-----+-----+-----+-----+
move.l #n,dx -> moveq #m,dx      | Y | Y | ? | * | ? | 2
      bchg.l dx,dx  | |  | |  | |
-----+-----+-----+-----+-----+
n = -32881 -> m = -113
n = -32849 -> m = -81
n = -32817 -> m = -49
n = -32785 -> m = -17
n = -16498 -> m = -114
n = -16466 -> m = -82
n = -16434 -> m = -50
n = -16402 -> m = -18
n = -8307 -> m = -115
n = -8275 -> m = -83
n = -8243 -> m = -51
n = -8211 -> m = -19
n = -4212 -> m = -116
n = -4180 -> m = -84
n = -4148 -> m = -52
n = -4116 -> m = -20
n = -2165 -> m = -117
n = -2133 -> m = -85
n = -2101 -> m = -53
n = -2069 -> m = -21
n = -1142 -> m = -118
n = -1110 -> m = -86
n = -1078 -> m = -54
n = -1046 -> m = -22
n = -631 -> m = -119
n = -599 -> m = -87
n = -567 -> m = -55

```

n = -535 -> m = -23  
 n = -376 -> m = -120  
 n = -344 -> m = -88  
 n = -312 -> m = -56  
 n = -280 -> m = -24  
 n = 264 -> m = 8  
 n = 296 -> m = 40  
 n = 328 -> m = 72  
 n = 360 -> m = 104  
 n = 521 -> m = 9  
 n = 553 -> m = 41  
 n = 585 -> m = 73  
 n = 617 -> m = 105  
 n = 1034 -> m = 10  
 n = 1066 -> m = 42  
 n = 1098 -> m = 74  
 n = 1130 -> m = 106  
 n = 2059 -> m = 11  
 n = 2091 -> m = 43  
 n = 2123 -> m = 75  
 n = 2155 -> m = 107  
 n = 4108 -> m = 12  
 n = 4140 -> m = 44  
 n = 4172 -> m = 76  
 n = 4204 -> m = 108  
 n = 8205 -> m = 13  
 n = 8237 -> m = 45  
 n = 8269 -> m = 77  
 n = 8301 -> m = 109  
 n = 16398 -> m = 14  
 n = 16430 -> m = 46  
 n = 16462 -> m = 78  
 n = 16494 -> m = 110  
 n = 32783 -> m = 15  
 n = 32815 -> m = 47  
 n = 32847 -> m = 79  
 n = 32879 -> m = 111

move.l #n,dx -> moveq #m,dx		N		N		?		*		?		2
bchg.l dx,dx												

n = -2147483617 -> m = 31  
 n = -2147483585 -> m = 63  
 n = -2147483553 -> m = 95  
 n = -2147483521 -> m = 127  
 n = -1073741922 -> m = -98  
 n = -1073741890 -> m = -66  
 n = -1073741858 -> m = -34  
 n = -1073741826 -> m = -2  
 n = -536871011 -> m = -99  
 n = -536870979 -> m = -67  
 n = -536870947 -> m = -35  
 n = -536870915 -> m = -3  
 n = -268435556 -> m = -100  
 n = -268435524 -> m = -68  
 n = -268435492 -> m = -36  
 n = -268435460 -> m = -4

n = -134217829 -> m = -101  
n = -134217797 -> m = -69  
n = -134217765 -> m = -37  
n = -134217733 -> m = -5  
n = -67108966 -> m = -102  
n = -67108934 -> m = -70  
n = -67108902 -> m = -38  
n = -67108870 -> m = -6  
n = -33554535 -> m = -103  
n = -33554503 -> m = -71  
n = -33554471 -> m = -39  
n = -33554439 -> m = -7  
n = -16777320 -> m = -104  
n = -16777288 -> m = -72  
n = -16777256 -> m = -40  
n = -16777224 -> m = -8  
n = -8388713 -> m = -105  
n = -8388681 -> m = -73  
n = -8388649 -> m = -41  
n = -8388617 -> m = -9  
n = -4194410 -> m = -106  
n = -4194378 -> m = -74  
n = -4194346 -> m = -42  
n = -4194314 -> m = -10  
n = -2097259 -> m = -107  
n = -2097227 -> m = -75  
n = -2097195 -> m = -43  
n = -2097163 -> m = -11  
n = -1048684 -> m = -108  
n = -1048652 -> m = -76  
n = -1048620 -> m = -44  
n = -1048588 -> m = -12  
n = -524397 -> m = -109  
n = -524365 -> m = -77  
n = -524333 -> m = -45  
n = -524301 -> m = -13  
n = -262254 -> m = -110  
n = -262222 -> m = -78  
n = -262190 -> m = -46  
n = -262158 -> m = -14  
n = -131183 -> m = -111  
n = -131151 -> m = -79  
n = -131119 -> m = -47  
n = -131087 -> m = -15  
n = -65648 -> m = -112  
n = -65616 -> m = -80  
n = -65584 -> m = -48  
n = -65552 -> m = -16  
n = 65552 -> m = 16  
n = 65584 -> m = 48  
n = 65616 -> m = 80  
n = 65648 -> m = 112  
n = 131089 -> m = 17  
n = 131121 -> m = 49  
n = 131153 -> m = 81  
n = 131185 -> m = 113  
n = 262162 -> m = 18

---

```

n = 262194 -> m = 50
n = 262226 -> m = 82
n = 262258 -> m = 114
n = 524307 -> m = 19
n = 524339 -> m = 51
n = 524371 -> m = 83
n = 524403 -> m = 115
n = 1048596 -> m = 20
n = 1048628 -> m = 52
n = 1048660 -> m = 84
n = 1048692 -> m = 116
n = 2097173 -> m = 21
n = 2097205 -> m = 53
n = 2097237 -> m = 85
n = 2097269 -> m = 117
n = 4194326 -> m = 22
n = 4194358 -> m = 54
n = 4194390 -> m = 86
n = 4194422 -> m = 118
n = 8388631 -> m = 23
n = 8388663 -> m = 55
n = 8388695 -> m = 87
n = 8388727 -> m = 119
n = 16777240 -> m = 24
n = 16777272 -> m = 56
n = 16777304 -> m = 88
n = 16777336 -> m = 120
n = 33554457 -> m = 25
n = 33554489 -> m = 57
n = 33554521 -> m = 89
n = 33554553 -> m = 121
n = 67108890 -> m = 26
n = 67108922 -> m = 58
n = 67108954 -> m = 90
n = 67108986 -> m = 122
n = 134217755 -> m = 27
n = 134217787 -> m = 59
n = 134217819 -> m = 91
n = 134217851 -> m = 123
n = 268435484 -> m = 28
n = 268435516 -> m = 60
n = 268435548 -> m = 92
n = 268435580 -> m = 124
n = 536870941 -> m = 29
n = 536870973 -> m = 61
n = 536871005 -> m = 93
n = 536871037 -> m = 125
n = 1073741854 -> m = 30
n = 1073741886 -> m = 62
n = 1073741918 -> m = 94
n = 1073741950 -> m = 126
n = 2147483551 -> m = -97
n = 2147483583 -> m = -65
n = 2147483615 -> m = -33
n = 2147483647 -> m = -1

```

```

-----+-----+-----+-----+-----+-----+-----+-----
move.l #n,m? -> moveq #n,ds      | Y | Y | ? | ? | ? | 2

```



```

asl.l #2,az      | | | |
add.l ax,az      | | | |
add.x #n,az      | | | |
-----+-----+-----+-----+-----+
az=n+ax+4*ay, -32767<=n<=32767
-----+-----+-----+-----+-----+
move.l ay,az -> lea n(ax,ay.l*8),az | - | - | ? | ? | ? | ?
asl.l #3,az      | | | |
add.l ax,az      | | | |
add.x #n,az      | | | |
-----+-----+-----+-----+-----+
az=n+ax+8*ay, -32767<=n<=32767
-----+-----+-----+-----+-----+
move.w #x,mn -> move.l #xy,mn      | Y | Y | ? | ? | ? | 2/4/6
move.w #y,mn+2      | | | |
-----+-----+-----+-----+-----+
best if mn is longword aligned
-----+-----+-----+-----+-----+
move.x #0,ax -> sub.l ax,ax      | Y | Y | ? | ? | ? | 2/4
-----+-----+-----+-----+-----+
move.x #n,ax -> lea n,ax      | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+
n <> 0
-----+-----+-----+-----+-----+
move.x (rx,ay),az -> move.x ay,az | Y | Y | ? | ? | ? | 0
add.x rx,az      | | | |
-----+-----+-----+-----+-----+
move.x ax,ay -> lea n(ax),ay      | Y | Y | ? | ? | ? | 2/4
add.x #n,ay      | | | |
-----+-----+-----+-----+-----+
-32767 <= n <= 32767
-----+-----+-----+-----+-----+
move.x ax,az -> lea -n(ax,ay),az | Y | Y | ? | ? | ? | 2
sub.x #n,az      | | | |
add.x ay,az      | | | |
-----+-----+-----+-----+-----+
az=n+ax+ay, n<=32767
-----+-----+-----+-----+-----+
move.x ax,az -> lea n(ax,ay),az | Y | Y | ? | ? | ? | 2
add.x #n,az      | | | |
add.x ay,az      | | | |
-----+-----+-----+-----+-----+
az=n+ax+ay, n<=32767
-----+-----+-----+-----+-----+
movem.l (ax)+,registers      | * | * | ? | ? | Y | *
-> move.l (ax)+,ry | | | |
for each reg | | | |
-----+-----+-----+-----+-----+
movem.w *,dx -> move.w *,dx      | Y | Y | ? | ? | ? | 0
ext.l dx      | | | |
-----+-----+-----+-----+-----+
movem.x *,@ -> move.x *,@      | Y | Y | ? | ? | ? | 2
| | | |
-----+-----+-----+-----+-----+
@ = a single register, not (@=dx & .x=.w)
-----+-----+-----+-----+-----+
movem.x @,* -> move.x @,*      | Y | Y | ? | ? | ? | 2

```

| | | |

-----+-----+-----+-----+-----  
 @ = a single register, status flags are wrong  
 -----+-----+-----+-----+-----

moveq #n,az -> lea n(ax,ay.1\*2),az | - | - | ? | ? | ? | ?  
 add.x ay,az | | | | | |  
 add.x ax,az | | | | | |  
 add.x ay,az | | | | | |  
 -----+-----+-----+-----+-----

az=n+ax+2\*ay, -128<=n<=127  
 -----+-----+-----+-----+-----

mul\*.l #1,dx -> (nothing) | ! | ! | Y | Y | Y | 6  
 -----+-----+-----+-----+-----

delete  
 -----+-----+-----+-----+-----

mul\*.l #10,dx -> add.l dx,dx | ! | ! | ? | ? | ? | -2  
 move.l dx,ds | | | | | |  
 asl.l #2,dx | | | | | |  
 add.l ds,dx | | | | | |  
 -----+-----+-----+-----+-----

mul\*.l #12,dx -> asl.l #2,dx | ! | ! | ? | ? | ? | -2  
 move.l dx,ds | | | | | |  
 add.l dx,dx | | | | | |  
 add.l ds,dx | | | | | |  
 -----+-----+-----+-----+-----

mul\*.l #2,dx -> add.l dx,dx | ! | ! | ? | ? | ? | 4  
 -----+-----+-----+-----+-----

mul\*.l #3,dx -> move.l dx,ds | ! | ! | ? | ? | ? | 0  
 add.l dx,dx | | | | | |  
 add.l ds,dx | | | | | |  
 -----+-----+-----+-----+-----

mul\*.l #5,dx -> move.l dx,ds | ! | ! | ? | ? | ? | 0  
 asl.l #2,dx | | | | | |  
 add.l ds,dx | | | | | |  
 -----+-----+-----+-----+-----

mul\*.l #6,dx -> add.l dx,dx | ! | ! | ? | ? | ? | -2  
 move.l dx,ds | | | | | |  
 add.l dx,dx | | | | | |  
 add.l ds,dx | | | | | |  
 -----+-----+-----+-----+-----

mul\*.l #7,dx -> move.l dx,ds | ! | ! | ? | ? | ? | 0  
 asl.l #3,dx | | | | | |  
 sub.l ds,dx | | | | | |  
 -----+-----+-----+-----+-----

mul\*.l #9,dx -> move.l dx,ds | ! | ! | ? | ? | ? | 0  
 asl.l #3,dx | | | | | |  
 add.l ds,dx | | | | | |  
 -----+-----+-----+-----+-----

mul\*.l #n,dx -> moveq #m,ds | ! | ! | ? | ? | ? | 2  
 asl.l ds,dx | | | | | |  
 -----+-----+-----+-----+-----

n is 2^m, 8<m<14  
 -----+-----+-----+-----+-----

muls.l #0,dx -> moveq #0,dx | ! | ! | ? | ? | ? | 4  
 -----+-----+-----+-----+-----

muls.l #n,dx -> asl.l #m,dx | ! | ! | ? | ? | ? | 4  
 -----+-----+-----+-----+-----



```

n is 2^m, 1 <= m <= 8
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
muls.w #n,dx -> moveq #m,ds      | Y | Y | ? | ? | ? | -2
  ext.l dx      | | | | |
  asl.l ds,dx   | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
n is 2^m, 8<m<14
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
muls.w #n,dx -> swap dx          | Y | Y | ? | ? | ? | -2
  clr.w dx      | | | | |
  asr.l #(16-m),dx | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
n is 2^m, 8 <= m <= 15
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.l #0,dx -> moveq #0,dx      | ! | ! | ? | ? | ? | 4
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.l #n,dx -> lsl.l #m,dx     | ! | ! | ? | ? | ? | 4
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
n is 2^m, 1 <= m <= ?
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #0,dx -> moveq #0,dx      | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #1,dx -> swap dx          | Y | Y | ? | ? | ? | -2
  clr.w dx      | | | | |
  swap dx       | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #12,dx -> swap dx         | Y | Y | ? | ? | ? | -10
  clr.w dx      | | | | |
  swap dx       | | | | |
  asl.l #2,dx   | | | | |
  move.l dx,ds  | | | | |
  add.l dx,dx   | | | | |
  add.l ds,dx   | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #2,dx -> swap dx          | Y | Y | ? | ? | ? | -4
  clr.w dx      | | | | |
  swap dx       | | | | |
  add.l dx,dx   | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #3,dx -> swap dx          | Y | Y | ? | ? | ? | -8
  clr.w dx      | | | | |
  swap dx       | | | | |
  move.l dx,ds  | | | | |
  add.l dx,dx   | | | | |
  add.l ds,dx   | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #5,dx -> swap dx          | Y | Y | ? | ? | ? | -8
  clr.w dx      | | | | |
  swap dx       | | | | |
  move.l dx,ds  | | | | |
  asl.l #2,dx   | | | | |
  add.l ds,dx   | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #6,dx -> swap dx          | Y | Y | ? | ? | ? | -10
  clr.w dx      | | | | |
  swap dx       | | | | |
  add.l dx,dx   | | | | |

```

```

    move.l dx,ds      | | | | |
    add.l ds,dx      | | | | |
    add.l ds,dx      | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #7,dx -> swap dx      | Y | Y | ? | ? | ? | -8
    clr.w dx        | | | | |
    swap dx         | | | | |
    move.l dx,ds    | | | | |
    asl.l #3,dx     | | | | |
    sub.l ds,dx     | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #9,dx -> swap dx      | Y | Y | ? | ? | ? | -8
    clr.w dx        | | | | |
    swap dx         | | | | |
    move.l dx,ds    | | | | |
    asl.l #3,dx     | | | | |
    add.l ds,dx     | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #n,dx -> swap dx      | Y | Y | ? | ? | ? | -4
    clr.w dx        | | | | |
    swap dx         | | | | |
    lsl.l #m,dx     | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+
    n is 2^m, 1 <= m <= 8
-----+-----+-----+-----+-----+-----+-----+-----+-----+
mulu.w #n,dx -> swap dx      | Y | Y | ? | ? | ? | -2
    clr.w dx        | | | | |
    lsr.l #(16-m),dx | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+
    n is 2^m, 8 <= m <= 15
-----+-----+-----+-----+-----+-----+-----+-----+-----+
neg.x dx    -> add.x dx,dy    | Y | Y | Y | ? | ? | 2
sub.x dx,dy | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+
    dx is trashed
-----+-----+-----+-----+-----+-----+-----+-----+-----+
neg.x dx    -> eor.x #n-1,dx  | Y | Y | ? | ? | ? | 2
add.x #n,dx | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+
    n is 2^m, dx < n
-----+-----+-----+-----+-----+-----+-----+-----+-----+
neg.x dx    -> sub.x dx,dy    | Y | Y | Y | ? | ? | 2
add.x dx,dy | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+
    dx is trashed
-----+-----+-----+-----+-----+-----+-----+-----+-----+
nop -> (nothing)      | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----+-----+-----+-----+
    remove nops
-----+-----+-----+-----+-----+-----+-----+-----+-----+
or.l #n,dx -> bset.l #b,dx   | Y | Y | ? | ? | ? | 2
-----+-----+-----+-----+-----+-----+-----+-----+-----+
    n = 2^b (only 1 bit set)
-----+-----+-----+-----+-----+-----+-----+-----+-----+
sub*.x #0,dx -> tst.x dx     | Y | Y | ? | ? | ? | 2/4
-----+-----+-----+-----+-----+-----+-----+-----+-----+
sub.x #n,* -> addq.x #-n,*   | Y | Y | ? | ? | ? | 2/4

```

```

-----+-----+-----+-----+-----+-----
-8 <= n <= -1
-----+-----+-----+-----+-----+-----
sub.x #n,* -> subq.x #n,*      | Y | Y | ? | ? | ? | 2/4
-----+-----+-----+-----+-----+-----
if 1 <= n <= 8
-----+-----+-----+-----+-----+-----
sub.x #n,ax -> lea -n(ax),ax   | Y | Y | ? | ? | ? | 0/2
-----+-----+-----+-----+-----+-----
-32767 <= n <= -9, 9 <= n <= 32767
-----+-----+-----+-----+-----+-----
subq.l #n,ax -> subq.w #n,ax  | Y | Y | ? | ? | ? | 0
-----+-----+-----+-----+-----+-----
subq.w #1,dx -> db<cc> dx,??  | y | y | ? | ? | ? | -2
b<cc> ?? b<cc> ??           | | | | | |
-----+-----+-----+-----+-----+-----
if dx=0 then will be slower
-----+-----+-----+-----+-----+-----
subq.w #1,dx -> dbf dx,??     | Y | Y | ? | ? | ? | -2
bra ?? bra ??               | | | | | |
-----+-----+-----+-----+-----+-----
if dx=0 then will be slower
-----+-----+-----+-----+-----+-----
tst.w dx -> dbra dx,??       | y | y | ? | ? | ? | 2
bne ??                       | | | | | |
-----+-----+-----+-----+-----+-----
dx will be trashed
-----+-----+-----+-----+-----+-----

```

---

#### H I N T S   &   T I P S

---

This new section is for stuff that cannot be included in the above tables. This can include pipelining optimizations and other stuff.

020+ Sequential memory accesses can cause pipeline stalls, so try and rearrange code so memory accesses do not immediately follow each other. The same problem occurs if an address register updated in one line is accessed in the next line.

ALL Include small routines as macros, because inline routines will be much faster, and in extreme cases smaller.

ALL If a subroutine is only called from one position, either move it inline, or only use jmp/bra commands.

---

#### C O N C L U S I O N

---

There are the optimizations i've come up with so far. If you could check what i've done, and report any errors, that would make this list better. I

---



-Z enable listing on pass 1. The information in the code file may be incorrect but this can be used to find mistakes when omitting an ENDC (OPT LIST1). This is provided for backwards compatibility; OPT TRACEIF can normally be used to find such errors more quickly.

---