MPE Forth 6 Cross Compiler

68xxx/683xx User Manual

MPE Forth 6 Cross Compiler

User manual

Manual revision 6.100

Date 10 July 2000

Software

Software version 6.100

Package Number:	

For technical support

Please contact your supplier

For further information

MicroProcessor Engineering Limited
133 Hill Lane, Southampton
SO15 5AF, UK
Tel: +44 (0)23 8063 1441
Fax: +44 (0)23 8033 9691

e-mail: mpe@mpeltd.demon.co.uk
tech-support@mpeltd.demon.co.uk

web: www.mpeltd.demon.co.uk

Acknowledgements

MPE would like to thank the following people for their involvement in the production of this product:

Steve Coul, Mark Davis, Stephen Pelc, Matthew Purvis

Licence terms

Distribution of application programs

Providing that the end user has no access to the underlying Forth and its text interpreter except for engineering and maintenance access only, applications compiled with the Forth 6 cross-compiler may be distributed without royalty. An acknowledgement will be gratefully appreciated. No part of the cross-compiler or the target source code may be further distributed without permission from MicroProcessor Engineering.

If you need to ship applications with an open Forth system, or wish to check what constitutes engineering and maintenance access, please contact MPE. An OEM version of ROM PowerForth is available for distribution with your products, and includes documentation on disc.

Warranties and support

We try to make our products as reliable and bug free as we possibly can. We support our products. If you find a bug in this product and its associated programs we will do our best to fix it. Please check first by fax or email to see if the problem has already been fixed. Please send us enough information including source code on disc or by email to us, so that we can replicate the problem and then fix it. Please also let us know the serial number of your system and its version number. We will then send you an update when we have fixed the problem. The level of technical support that we can offer may depend on the Support Policy bought with the product.

Technical support will only be available on the current version of the product.

Make as many copies as you need for backup and security. The issue discs or CD are not copy protected. The code is copyrighted material and only ONE copy of it should be use at any one time. Contact MPE or your vendor for details of multiple copy terms and site licensing.

As this copy is sold direct and through dealers and purchasing departments, we cannot keep track of all our users. If you fill out the registration form enclosed and send it back to us, we will put you on our mailing list. This way we will be able to keep you informed of updates and new extensions, as they become available. If you need technical support from us we will need these details in order to respond to you. You will find the serial number of the system on the original issue discs.

Contents

List of Tables

1 Introduction

What you get

The MPE Forth 6.1 cross compiler for the 68xxx/683xx family consists of five portions which will have been installed for you.

1) AIDE front end

2) Support tools - MAKE utility, Intel Hex and Motorola S-record converters

3) Forth 6 cross compiler for 68xxx/683xxx

4) Standalone Forth source code and configuration files
   Note that the standalone target is NOT supplied with the IRTC and Forth Stamp compilers.

5) Umbilical Forth source code and configuration files

The best reference for the target code is to read it yourself.

Enhancements from v6.0 to 6.1

The target code for v6.1 is faster than the v6.0 code because of improved code generation in the cross compiler, and more highly tuned code in the target. The target code is also smaller. Full details of the changes are available in the file DOCS\RELEASE.68K.

Target code disassembler. You can now disassemble any definition, whether Forth or code. You can also see some of the new code generation optimisations.

Generic I/O permits you to add new I/O devices very easily and to use the standard I/O words such as **`KEY`** and **`EMIT`** with them. Each task may access a different default device, and the default can be changed at any time.

TIMEBASE multiple slaved timer system.

More examples.

Supported CPUs

This family of CPUs is supplied by a large number of silicon vendors. The target code will run on any of the CPUs, and there are supplied UART drivers for several versions..

Standard hardware

The standard hardware boards supported by this compiler are:

- CMS 68307 MicroModule (M68307)
- Forth-Systeme 68332 Module (M68332)

2 VFX code generator

The VFX code generator makes the use of assembly code redundant except in the most extreme circumstances.

In the main the VFX code generator is a black box. There are however a few control switches which are useful in limited cases. The default conditions provide the best balance between speed and size.

VFX control switches

Forward branches such as those generated by **IF** and **WHILE** are now by default short branches with a range of +127 bytes giving smaller and faster code. If the optimiser reports an error, bracket the offending definition with the following pair of directives.

-SHORT-BRANCHES \ -- ; disable short forward branches

+SHORT-BRANCHES \ -- ; enable short forward branches

Note that the two directives above also affect the size of forward branches compiled by the assembler control structures. See the assembler chapter for more details of these structures.

By default, the entry code for **DO** and **?DO** is compiled inline, which is faster but a bit larger in some cases. Inline entry code is controlled by the following directives:

FAST-DO \ -- ; enable inline DO

-FAST-DO \ -- ; disable inline DO

FAST-DO? \ -- flag ; test inline DO state

The following directives are included as dummies for compatibility with the other v6.1 VFX compilers and the common target code.

+FASTLOOPS \ --

-FASTLOOPS \ --

FASTER \ --

SMALLER \ --

IS-SCRATCH \ n --

#SCRATCH	\ -- n
IS-LP	\ n --
IS-UP	\ n --

3 68xxx/683xx cross assembler

The MPE cross compiler has a built-in cross-assembler. This gives you the ability to define new Forth words in assembler as well as in Forth. You can also assemble code to anywhere in memory.

Why write in assembler?

Forth is compact and quick, so why write in assembler? An assembler definition is normally quicker than a group of corresponding forth words.

Creating Forth words in assembler

Forth words can easily be defined in assembler. They increase the execution speed of your code and can sometimes make your code smaller.

Defining assembler words

Forth words written in assembler follow a similar form to a word written in forth. Instead of a colon you have **CODE**. Instead of semi-colon you have **END-CODE**. For example:

```
CODE <name>
  ...
  ...
  NEXT,
END-CODE
```

creates a word called <name>. Any assembler code between the **CODE** and **END-CODE** will be assembled into the word. When executed, the command **NEXT**, will stop the execution of the assembler and return to the calling word.

Writing assembler words

The syntax used for the opcodes has been kept as similar to the Motorola syntax as possible. See the list at the end of the chapter for a comparison of Motorola versus Forth syntax.

Preserving the Forth registers

The Forth interpreter and compiler use some of the target processor's registers. These must be preserved if they are used in the assembler. They can be saved on the stack, in memory or in other registers and restored at the end of the word. The 68k registers that are used are shown in the table below.

68k register	Forth register	Function
A7	RSP	Forth return stack pointer, do not change this without good reason, this holds return addresses.
A6	PSP	Forth data stack pointer, do not change this without good reason, use it for passing parameters between words.
A5	UP	Pointer to the base of the current User Area.
A4	LP	Local variable frame pointer
A3		A constant value of 'zero'
D7	TOS	Instead of holding the top item of the Forth data stack in main memory, it is held in a register. This allows many simple operations to execute faster, and it also reduces the amount of memory traffic.
A0..A2, D0..D6		Scratch

Table 1: The Forth Registers

Executing an assembler word

A Forth word written in assembler is executed in the same way as a word written in Forth. It is executed in the same way as a normal word, by stating its name.

Assembling into memory

Assembler code can be assembled into memory and not in a Forth word. To do this you need to:

- turn on the assembler
- write your assembler code
- turn off the assembler

To turn on the assembler, use the word **ASMCODE**. To switch back to Forth use the word **END-CODE**. Between the **ASMCODE** and **END-CODE** definitions, any assembler will be assembled. The assembled code will be placed in the dictionary without a header. The code can be executed by the use of labels. This is often used to define low-level interrupts. See the chapter on Interrupts, for more details on writing low-level interrupts.

Creating defining words in assembler

The cross compiler allows you to define the run-time (**DOES>**) part of a defining word in assembler. To do this use **;CODE** in the form:

```
: <name>
  CREATE
    ...
    ...
  ;CODE
    ...
    ...
END-CODE
```

An example is shown below:

```
: VARIABLE      \ <spaces>name -- ; -- addr
  CREATE                    \ Create header
    0 ,                     \ Initial value
  ;CODE                     \ Run-time action
    d7 -(a6) move,          \ Save TOS
    (a7)+ a0 movea,         \ get PFA
    (a0) d7 move,           \ get variable addr.
    NEXT,
END-CODE
```

Structured programming

Three facilities are available to give you the advantages of structured programming, in assembler:

- control structures
- labels
- local labels

Control structures

There are assembler equivalents to the Forth control structures. The available structures are:

```
cc IF, ... THEN,
cc IF, ... ELSE, ... THEN,
BEGIN, ... cc UNTIL,
BEGIN, ... cc WHILE, ... REPEAT,
BEGIN, ... AGAIN,
```

where cc is one of the condition codes in the table below.

Code	Condition	Code	Condition
F,	Always false	EQ,	Equal
T,	Always true	NE,	Not equal
CS,	Carry Set	GE,	Greater than or equal
CC,	Carry Clear	LT,	Less than
PL,	Plus (+ve or 0)	GT,	Greater than
MI,	Minus (-ve)	LS,	Unsigned less than or equal
VS,	Overflow Set	HS,	Unsigned greater than or equal to
VC,	Overflow clear	LO,	Unsigned less than
LE,	Less than or equal to	HI,	Unsigned greater than

Table 2: Available condition codes

Labels

Labels can be used to mark a place in assembler code. That place can then be referenced in other areas of code.

Creating a label

Labels can be defined by using the command **`L:`** . It is used in the form:

```
l: <name>
```

where **`<name>`** is what you want to call the label.

Referencing a label

A label is referenced by stating its name. For example,

```
<name> BEQ,
```

will assemble to `branch if equal to **`<name>`**'.

Local labels

If you need to use labels within a code definition, you may use the local labels provided. These are used just as normal labels in the assembler, but some restrictions apply:

- there is a maximum of ten labels
- the names are in the form L$n where n is in the range 1 to 10
- a reference is valid until the next occurrence of **CODE** or **;CODE**

Creating a local label

To define a local label use **L$n:**, where n is a number from one to ten. For example:

```
L$1:
```

Referencing a local label

To reference a local label, type its name. For example,

```
L$1 BRA,
```

assembles code for a branch to **L$1:**.

Creating macros

A macro is a word that lays down code `in-line' within an assembler definition. They are normally used when there is a repetitive use of a series of opcodes.

Defining a macro

A macro is defined using colon and semi-colon. It must also be defined in the cross compiler's vocabulary, **ASM-ACCESS**. The place to create a macro is in the control file and it must be defined before the word **CROSS-COMPILE**. As an example, the macro **NEXT**, is shown below. **NEXT**, is defined as a macro, so each time it is used, its code is layed down. This makes it quicker than calling a subroutine.

```
\ switch to cross compilers assembler vocab
FORTH ALSO C-C ALSO ASSEMBLER
ALSO ASM-ACCESS DEFINITIONS

\ define NEXT
: NEXT,\ -- lay in-line next code
  rts,
;

\ switch back to normal forth vocabulary
ONLY FORTH DEFINITIONS
```

Using a macro

A macro is used by stating its name. For example, in a **CODE** definition, **NEXT**, is a macro.`

Debugging

It is possible to disassemble compiled words using the word **XDASM <name>**.

This can be done during compilation by including an **XDASM** statement in the control file, or interactively after compilation by including the word **INTERACTIVE** before the **FINIS**.

Addressing modes

The 68xxx family has a large range of addressing modes. They are described here by comparing the Forth syntax with the conventional syntax. For instructions which have two operands (source and destination), the operands are separated by spaces, not commas. Opcodes end with a comma to prevent conflict between Forth words and 68000 opcode names (e.g. **MOVE** and **SWAP**).

Do not forget that the 68000 uses SIGNED address arithmetic, and that if a 16-bit word is moved, added, or subtracted into an address register, the data in the low word is sign extended into the high word.

Notation

All numbers entered to Forth, whether data or addresses, can be entered as single precision 32-bit numbers. In the text of this manual, but NOT in the source code, hexadecimal numbers are often shown in the form:

0xx.yyyyh

for example

010.80FFh

This is done to clarify the presentation of 24- or 32-bit numbers with 6 or 8 hexadecimal digits. The `.' character indicates the split between the high and low 16-bit halves of the 32-bit word. It is not required in the source code, and will cause an error if it is present.

Specifying data size

The 68000 can operate on byte, word (16-bit), or long word (32-bit) data. If no other indication is given, this assembler defaults to LONG operations to match the 32-bit stack size.

To specify size, use **B. W.** or **L.** before the opcode, e.g:

```
D0 (A5) L. MOVE,
```

replaces the conventional:

```
MOVE.L D0,(A5)
```

Register addressing

The registers are defined as **D0..D7** and **A0..A7**, for example to move **D0** into **A6** use:

```
D0 A6 L. MOVEA,
```

instead of:

```
MOVEA.LD0,A6
```

Immediate addressing

This mode is signified by the **#** (hash) symbol. Note than in the U.K, there is often confusion on printers between the hash symbol and the pound symbol. To add a value to **D3** use:

HEX

```
# 01066 D3 W. ADD,
```

instead of:

```
ADD.W  #$1066,D3
```

The use of the dollar **$** symbol is changed to represent absolute short (direct) addressing, and the base must be indicated in the usual way using **BINARY OCTAL DECIMAL** and **HEX**.

Absolute short addressing

To move a value to or from a location with short addressing (addresses 00.0000h to 00.7FFFh and 0FF.8000h to 0FF.FFFFh) use:

```
(A7)+ $ 300 W. MOVE,
```

instead of:

```
MOVE.W (A7)+,0300
```

Absolute long addressing

This mode is indicated by **$L**.

```
(A7)+ $L 303000 W. MOVE,
```

instead of

```
MOVE.W (A7)+,303000
```

Address register indirect addressing

The register used may be any of the address registers, using the Forth form:

```
(A3) D3 L. ADD,
```

instead of:

```
MOVE.L (A3),D3
```

Auto-increment & auto-decrement modes

These modes are indicated by preceding the address indirect form with a leading **-** for pre-decrement, or suffixing it with a + for post-increment. Use the Forth forms:

```
(A3)+ D3 L. ADD,
D0 -(A7) L. MOVE,
```

instead of:

```
ADD.L  (A3)+,D3
MOVE.L D0,-(A7)
```

Address register indirect with displacement

This mode adds a displacement to the address register, or PC relative may be indicated by using **d(PC)**. Use the Forth form:

```
56 d(A3) D3 W. MOVE,
```

instead of:

```
MOVE.W 56(A3),D3
```

Address indirect with index and displacement

This mode adds an index (address or data register, word or byte) and an offset to the address register. The address indirect register is indicated by:

```
d(A0, … d(A7,
```

Word index registers are indicated by:

```
A0) … A7)
D0) … D7)
```

Long index registers are indicated by:

```
A0.L) … A7.L)
D0.L) … D7.L)
```

Use the Forth form:

```
20 d(A3, D7.L) D3 B. ADD,
```

instead of:

```
ADD.B  20(A3,D7.L),D3
```

Zero operand instructions

Some instructions need no operands, so none are provided. For example use the Forth form:

```
RTS,
```

instead of:

```
RTS
```

Register lists for MOVEM instructions

The MOVEM instruction requires a register list as one of its operands. The list is opened and closed by the braces **{** and **}** and the contents of the list are single registers, or register sets such as **D0-3** or **A1-4**. For example the Forth registers can be pushed and popped by means of:

```
{ D7 A3-6 } -(A7) L. MOVEM,
(A7)+ { D7 A3-6 } L. MOVEM,
```

instead of:

```
MOVEM.L D7/A3-6,-(A7)
MOVEM.L (A7)+,D7/A3-6
```

CPU selection

The instruction set of the processor is extended on various processor cores. The selection available in this assembler is for the 68000 (the base set, as used by the 68EC000 and some 6830x processors), the 68010, the CPU32 (based on the 68020 core and used by the 68332 and other 683xx processors) and the ColdFire. The following directives can be used to select the required instruction set. No operands are required.

```
M68000        \ -- ; select base 68000 instruction set
M68010        \ -- ; select 68010 instruction set
CPU32         \ -- ; select CPU32 instruction set
```

The following directives can be used to control conditional compilation of code according to the CPU in use, for example where CPU32 processors can take advantage of the extended multiply and divide capabilities.

```
M68000?       \ -- flag ; true if CPU is 68000 or above
M68010?       \ -- flag ; true if CPU is 68010 or above
CPU32?        \ -- flag ; true if CPU is CPU32 or above
```

Instructions by grouping

The 68xxx instructions are handled regularly in the main, except that those instructions that have special forms to deal with the condition code register or the user/supervisor stack pointer, have special forms. Instructions using condition codes (such as Bcc and Scc) have the condition code indicated by `cc', and any of the codes may be used.

Standard forms

The following instructions are treated and used regularly. Their special cases are detailed in the next section.

```
ABCD, ADD, ADDA, ADDI, ADDQ, ADDX, AND, ANDI, ASL, ASR,
Bcc, BCHG, BCLR, BRA, BSET, BSR, BTST,
CHK, CLR, CMP, CMPA, CMPI, CMPM,
DBcc, DIVS, DIVU,
EOR, EORI, EXG, EXT,
JMP, JSR,
LEA, LINK, LSL, LSR,
MOVE, MOVEA, MOVEM, MOVEP, MOVEQ, MULS, MULU,
NBCD, NEG, NEGX, NOP, NOT,
OR, ORI,
PEA,
RESET, ROL, ROR, ROXL, ROXR, RTE, RTR, RTS,
SBCD, Scc, STOP, SUB, SUBA, SUBI, SUBQ, SUBX, SWAP,
TAS, TRAP, TRAPV, TST,
UNLK,
```

Special cases

```
ANDSI,        AND immediate with the status register.
EORSI,        Exclusive OR immediate with status reg.
MOVE>>CCR,    Move to condition code register.
MOVE>>SR,     Move to status register
MOVE<<SR,     Move from status register
MOVE>>USP,    Move to user stack pointer
MOVE<<USP,    Move from user stack pointer
```

Number bases

The number base in the Forth assembler can be indicated by the words **`BINARY`** **`DECIMAL`** and **`HEX`**. In addition, numbers prefixed by the `$' `#' and `%' characters are treated as special cases. These characters affect the number base for that number only. Note that the characters `$' and `%' follow Motorola usage. Note that the `#' symbol attached to a number is not the same as the # word that indicates immediate addressing.

Symbol	Base	Example
$	hex	$55AA
#	decimal	#1234
%	binary	%1011001
@	octal	@454

Table 3: Number bases

Instruction syntax

The instructions are shown alphabetically with all their addressing forms.

Notation

The notation follows the Motorola conventions and is shown in the following table.

Text	Mode	Text	Mode
Dn	Data register	An	Address register
I	Immediate value	d	Absolute (short) address
dl	Absolute (long) address		

Table 4: 68xxx instruction notation

Instruction list

Conventional	Forth
abcd.b Dn,Dn	Dn Dn b. abcd,
abcd.b -(An),-(An)	-(An) -(An) b. abcd,
abcd.w Dn,Dn	Dn Dn w. abcd,
abcd.w -(An),-(An)	-(An) -(An) w. abcd,
abcd.l Dn,Dn	Dn Dn l. abcd,
abcd.l -(An),-(An)	-(An) -(An) l. abcd,
add.b Dn,Dn	Dn Dn b. add,
add.b (An),Dn	(An) Dn b. add,
add.b -(An),Dn	-(An) Dn b. add,
add.b (An)+,Dn	(An)+ Dn b. add,
add.b d(An),Dn	n d(An) Dn b. add,
add.b d(An,Dn.l),Dn	n d(An, Dn.l) Dn b. add,
add.b $d,Dn	$ n Dn b. add,
add.b $dl,Dn	$L n Dn b. add,
add.b $var,Dn	$ var Dn b. add,
add.b #i,Dn	# n Dn b. add,
add.w Dn,Dn	Dn Dn w. add,
add.w (An),Dn	(An) Dn w. add,
add.w -(An),Dn	-(An) Dn w. add,
add.w (An)+,Dn	(An)+ Dn w. add,
add.w d(An),Dn	n d(An) Dn w. add,
add.w d(An,Dn.l),Dn	n d(An, Dn.l) Dn w. add,
add.w $d,Dn	$ n Dn w. add,
add.w $dl,Dn	$L n Dn w. add,
add.w $var,Dn	$ var Dn w. add,
add.w #i,Dn	# n Dn w. add,
add.l Dn,Dn	Dn Dn l. add,
add.l (An),Dn	(An) Dn l. add,
add.l -(An),Dn	-(An) Dn l. add,
add.l (An)+,Dn	(An)+ Dn l. add,
add.l d(An),Dn	n d(An) Dn l. add,
add.l d(An,Dn.l),Dn	n d(An, Dn.l) Dn l. add,
add.l $d,Dn	$ n Dn l. add,
add.l $dl,Dn	$L n Dn l. add,
add.l $var,Dn	$ var Dn l. add,
add.l #i,Dn	# n Dn l. add,
add.b Dn,(An)	2 (An) b. add,
add.b Dn,-(An)	Dn -(An) b. add,
add.b Dn,(An)+	Dn (An)+ b. add,
add.b Dn,d(An)	Dn n d(An) b. add,
add.b Dn,d(An,Dn.l)	Dn n d(An, Dn.l) b. add,
add.b Dn,$d	Dn $ n b. add,
add.b Dn,$dl	Dn $L n b. add,
add.b Dn,$var	Dn $ var b. add,

```
add.w Dn,(An)              Dn (An) w. add,
add.w Dn,-(An)             Dn -(An) w. add,
add.w Dn,(An)+             Dn (An)+ w. add,
add.w Dn,d(An)             Dn n d(An) w. add,
add.w Dn,d(An Dn.l)        Dn n d(An, Dn.l) w. add,
add.w Dn,$d                Dn $ n w. add,
add.w Dn,$dl               Dn $L n w. add,
add.w Dn,$var              Dn $ var w. add,
add.l Dn,(An)              Dn (An) l. add,
add.l Dn,-(An)             Dn -(An) l. add,
add.l Dn,(An)+             Dn (An)+ l. add,
add.l Dn,d(An)             Dn n d(An) l. add,
add.l Dn,d(An,Dn.l)        Dn n d(An, Dn.l) l. add,
add.l Dn,$d                Dn $ n l. add,
add.l Dn,$dl               Dn $L n l. add,
add.l Dn,$var              Dn $ var l. add,

adda.w An,An               An An w. adda,
adda.w (An),An             (An) An w. adda,
adda.w -(An),An            -(An) An w. adda,
adda.w (An)+,An            (An)+ An w. adda,
adda.w d(An),An            n d(An) An w. adda,
adda.w d(An,Dn.l),An       n d(An, Dn.l) An w. adda,
adda.w $d,An               $ n An w. adda,
adda.w $dl,An              $L n An w. adda,
adda.w $var,An             $ var An w. adda,
adda.w #i,An               # n An w. adda,
adda.l An,An               An An l. adda,
adda.l (An),An             (An) An l. adda,
adda.l -(An),An            -(An) An l. adda,
adda.l (An)+,An            (An)+ An l. adda,
adda.l d(An),An            n d(An) An l. adda,
adda.l d(An,Dn.l),An       n d(An, Dn.l) An l. adda,
adda.l $d,An               $ n An l. adda,
adda.l $dl,An              $L n An l. adda,
adda.l $var,An             $ var An l. adda,
adda.l #i,An               # n An l. adda,

addi.b #i,(An)             # n (An) b.addi,
addi.b #i,-(An)            # n -(An) b. addi,
addi.b #i,(An)+            # n (An)+ b. addi,
addi.b #i,d(An)            # n n d(An) b. addi,
addi.b #i,d(An,Dn.l)       # n n d(An, Dn.l) b. addi,
addi.b #i,$d               # n $ n b. addi,
addi.b #i,$dl              # n $L n b. addi,
addi.b #i,$var             # n $ var b. addi,
addi.w #i,(An)             # n (An) w. addi,
addi.w #i,-(An)            # n -(An) w. addi,
```

```
addi.w #i,(An)+             # n (An)+ w. addi,
addi.w #i,d(An)             # n n d(An) w. addi,
addi.w #i,d(An,Dn.l)        # n n d(An, Dn.l) w. addi,
addi.w #i,$d                # n $ n w. addi,
addi.w #i,$dl               # n $L n w. addi,
addi.w #i,$var              # n $ var w. addi,
addi.l #i,(An)              # n (An) l. addi,
addi.l #i,-(An)             # n -(An) l. addi,
addi.l #i,(An)+             # n (An)+ l. addi,
addi.l #i,d(An)             # n n d(An) l. addi,
addi.l #i,d(An,Dn.l)        # n n d(An, Dn.l) l. addi,
addi.l #i,$d                # n $ n l. addi,
addi.l #i,$dl               # n $L n l. addi,
addi.l #i,$var              # n $ var l. addi,

addq.b #1,(An)              # n (An) b. addq,
addq.b #1,-(An)             # n -(An) b. addq,
addq.b #1,(An)+             # n (An)+ b. addq,
addq.b #1,d(An)             # n n d(An) b. addq,
addq.b #1,d(An,Dn.l)        # n n d(An, Dn.l) b. addq,
addq.b #1,$d                # n $ n b. addq,
addq.b #1,$dl               # n $L n b. addq,
addq.b #1,$var              # n $ var b. addq,
addq.w #1,(An)              # n (An) w. addq,
addq.w #1,-(An)             # n -(An) w. addq,
addq.w #1,(An)+             # n (An)+ w. addq,
addq.w #1,d(An)             # n n d(An) w. addq,
addq.w #1,d(An,Dn.l)        # n n d(An, Dn.l) w. addq,
addq.w #1,$d                # n $ n w. addq,
addq.w #1,$dl               # n $L n w. addq,
addq.w #1,$var              # n $ var w. addq,
addq.l #1,(An)              # n (An) l. addq,
addq.l #1,-(An)             # n -(An) l. addq,
addq.l #1,(An)+             # n (An)+ l. addq,
addq.l #1,d(An)             # n n d(An) l. addq,
addq.l #1,d(An,Dn.l)        # n n d(An, Dn.l) l. addq,
addq.l #1,$d                # n $ n l. addq,
addq.l #1,$dl               # n $L n l. addq,
addq.l #1,$var              # n $ var l. addq,

addx.b Dn,Dn                Dn Dn b. addx,
addx.w Dn,Dn                Dn Dn w. addx,
addx.l Dn,Dn                Dn Dn l. addx,

addx.b -(An),-(An)          -(An) -(An) b. addx,
addx.w -(An),-(An)          -(An) -(An) w. addx,
addx.l -(An),-(An)          -(An) -(An) l. addx,
```

```
and.b Dn,Dn                Dn Dn b. and,
and.b (An),Dn              (An) Dn b. and,
and.b -(An),Dn             -(An) Dn b. and,
and.b (An)+,Dn             (An)+ Dn b. and,
and.b d(An),Dn             n d(An) Dn b. and,
and.b d(An,Dn.l),Dn        n d(An, Dn.l) Dn b. and,
and.b $d,Dn                $ n Dn b. and,
and.b $dl,Dn               $L n Dn b. and,
and.b $var,Dn              $ var Dn b. and,
and.b #i,Dn                # n Dn b. and,
and.w Dn,Dn                Dn Dn w. and,
and.w (An),Dn              (An) Dn w. and,
and.w -(An),Dn             -(An) Dn w. and,
and.w (An)+,Dn             (An)+ Dn w. and,
and.w d(An),Dn             n d(An) Dn w. and,
and.w d(An,Dn.l),Dn        n d(An, Dn.l) Dn w. and,
and.w $d,Dn                $ n Dn w. and,
and.w $dl,Dn               $L n Dn w. and,
and.w $var,Dn              $ var Dn w. and,
and.w #i,Dn                # n Dn w. and,
and.l Dn,Dn                Dn Dn l. and,
and.l (An),Dn              (An) Dn l. and,
and.l -(An),Dn             -(An) Dn l. and,
and.l (An)+,Dn             (An)+ Dn l. and,
and.l d(An),Dn             n d(An) Dn l. and,
and.l d(An,Dn.l),Dn        n d(An, Dn.l) Dn l. and,
and.l $d,Dn                $ n Dn l. and,
and.l $dl,Dn               $L n Dn l. and,
and.l $var,Dn              $ var Dn l. and,
and.l #i,Dn                # n Dn l. and,
and.b Dn,(An)              Dn (An) b. and,
and.b Dn,-(An)             Dn -(An) b. and,
and.b Dn,(An)+             Dn (An)+ b. and,
and.b Dn,d(An)             Dn n d(An) b. and,
and.b Dn,d(An,Dn.l)        Dn n d(An, Dn.l) b. and,
and.b Dn,$d                Dn $ n b. and,
and.b Dn,$dl               Dn $L n b. and,
and.b Dn,$var              Dn $ var b. and,
and.w Dn,(An)              Dn (An) w. and,
and.w Dn,-(An)             Dn -(An) w. and,
and.w Dn,(An)+             Dn (An)+ w. and,
and.w Dn,d(An)             Dn n d(An) w. and,
and.w Dn,d(An Dn.l)        Dn n d(An, Dn.l) w. and,
and.w Dn,$d                Dn $ n w. and,
and.w Dn,$dl               Dn $L n w. and,
and.w Dn,$var              Dn $ var w. and,
and.l Dn,(An)              Dn (An) l. and,
and.l Dn,-(An)             Dn -(An) l. and,
```

```
and.l Dn,(An)+              Dn (An)+ l. and,
and.l Dn,d(An)              Dn n d(An) l. and,
and.l Dn,d(An,Dn.l)         Dn n d(An, Dn.l) l. and,
and.l Dn,$d                 Dn $ n l. and,
and.l Dn,$dl                Dn $L n l. and,
and.l Dn,$var               Dn $ var l. and,

andi.b #i,(An)              # n (An) b. andi,
andi.b #i,-(An)             # n -(An) b. andi,
andi.b #i,(An)+             # n (An)+ b. andi,
andi.b #i,d(An)             # n n d(An) b. andi,
andi.b #i,d(An,Dn.l)        # n n d(An, Dn.l) b. andi,
andi.b #i,$d                # n $ n b. andi,
andi.b #i,$dl               # n $L n b. andi,
andi.b #i,$var              # n $ var b. andi,
andi.w #i,(An)              # n (An) w. andi,
andi.w #i,-(An)             # n -(An) w. andi,
andi.w #i,(An)+             # n (An)+ w. andi,
andi.w #i,d(An)             # n n d(An) w. andi,
andi.w #i,d(An,Dn.l)        # n n d(An, Dn.l) w. andi,
andi.w #i,$d                # n $ n w. andi,
andi.w #i,$dl               # n $L n w. andi,
andi.w #i,$var              # n $ var w. andi,
andi.l #i,(An)              # n (An) l. andi,
andi.l #i,-(An)             # n -(An) l. andi,
andi.l #i,(An)+             # n (An)+ l. andi,
andi.l #i,d(An)             # n n d(An) l. andi,
andi.l #i,d(An,Dn.l)        # n n d(An, Dn.l) l. andi,
andi.l #i,$d                # n $ n l. andi,
andi.l #i,$dl               # n $L n l. andi,
andi.l #i,$var              # n $ var l. andi,

andi.b #i,ccr               # n ccr b. andi,

andi.w #i,sr                # n sr w. andi,

asl.b Dn,Dn                 Dn Dn b. asl,
asl.b #i,Dn                 # n Dn b. asl,
asl.w Dn,Dn                 Dn Dn w. asl,
asl.w #i,Dn                 # n Dn w. asl,
asl.l Dn,Dn                 Dn Dn l. asl,
asl.l #i,Dn                 # n Dn l. asl,

asl.b Dn                    # 1 Dn b. asl,
asl.b (An)                  # 1 (An) b. asl,
asl.b -(An)                 # 1 -(An) b. asl,
asl.b (An)+                 # n (An)+ b. asl,
asl.b d(An)                 # 1 n d(An) b. asl,
```

```
asl.b d(An,Dn.l)          # 1 n d(An, Dn.l) b. asl,
asl.b $d                  # 1 $ n b. asl,
asl.b $dl                 # 1 $L n b. asl,
asl.b $var                # 1 $ var b. asl,
asl.w Dn                  # 1 Dn w. asl,
asl.w (An)                # 1 (An) w. asl,
asl.w -(An)               # 1 -(An) w. asl,
asl.w (An)+               # 1 (An)+ w. asl,
asl.w d(An)               # 1 n d(An) w. asl,
asl.w d(An,Dn.l)          # 1 n d(An, Dn.l) w. asl,
asl.w $d                  # 1 $ n w. asl,
asl.w $dl                 # 1 $L n w. asl,
asl.w $var                # 1 $ var w. asl,
asl.l Dn                  # 1 Dn l. asl,
asl.l (An)                # 1 (An) l. asl,
asl.l -(An)               # 1 -(An) l. asl,
asl.l (An)+               # 1 (An)+ l. asl,
asl.l d(An)               # 1 n d(An) l. asl,
asl.l d(An,Dn.l)          # 1 n d(An, Dn.l) l. asl,
asl.l $d                  # 1 $ n l. asl,
asl.l $dl                 # 1 $L n l. asl,
asl.l $var                # 1 $ var l. asl,

asr.b Dn,Dn               Dn Dn b. asr,
asr.b #i,Dn               # n Dn b. asr,
asr.w Dn,Dn               Dn Dn w. asr,
asr.w #i,Dn               # n Dn w. asr,
asr.l Dn,Dn               Dn Dn l. asr,
asr.l #i,Dn               # n Dn l.asr,

asr.b Dn                  # 1 Dn b. asr,
asr.b (An)                # 1 (An) b. asr,
asr.b -(An)               # 1 -(An) b. asr,
asr.b (An)+               # 1 (An)+ b. asr,
asr.b d(An)               # 1 n d(An) b. asr,
asr.b d(An,Dn.l)          # 1 n d(An, Dn.l) b. asr,
asr.b $d                  # 1 $ n b. asr,
asr.b $dl                 # 1 $L n b. asr,
asr.b $var                # 1 $ var b. asr,
asr.w Dn                  # 1 Dn w. asr,
asr.w (An)                # 1 (An) w. asr,
asr.w -(An)               # 1 -(An) w. asr,
asr.w (An)+               # 1 (An)+ w. asr,
asr.w d(An)               # 1 n d(An) w. asr,
asr.w d(An,Dn.l)          # 1 n d(An, Dn.l) w. asr,
asr.w $d                  # 1 $ n w. asr,
asr.w $dl                 # 1 $L n w. asr,
asr.w $var                # 1 $ var w. asr,
```

```
asr.l Dn                    # 1 Dn l. asr,
asr.l (An)                  # 1 (An) l. asr,
asr.l -(An)                 # 1 -(An) l. asr,
asr.l (An)+                 # 1 (An)+ l. asr,
asr.l d(An)                 # 1 n d(An) l. asr,
asr.l d(An,Dn.l)            # 1 n d(An, Dn.l) l. asr,
asr.l $d                    # 1 $ n l. asr,
asr.l $dl                   # 1 $L n l. asr,
asr.l $var                  # 1 $ var l. asr,

bhi label                   label bhi,
bls label                   label bls,
bcc label                   label bcc,
bcs label                   label bcs,
bne label                   label bne,
beq label                   label beq,
bvc label                   label bvc,
bvs label                   label bvs,
bpl label                   label bpl,
bmi label                   label bmi,
bge label                   label bge,
blt label                   label blt,
bgt label                   label bgt,
ble label                   label ble,

bchg.b Dn,Dn                Dn Dnb. bchg,
bchg.b Dn,(An)              Dn (An) b. bchg,
bchg.b Dn,-(An)             Dn -(An) b. bchg,
bchg.b Dn,(An)+             Dn (An)+ b. bchg,
bchg.b Dn,d(An)             Dn n d(An) b. bchg,
bchg.b Dn,d(An,Dn.l)        Dn n d(An, Dn.l) b. bchg,
bchg.b Dn,$d                Dn $ n b. bchg,
bchg.b Dn,$var              Dn $ var b. bchg,
bchg.l Dn,Dn                Dn Dn l. bchg,
bchg.l Dn,(An)              Dn (An) l. bchg,
bchg.l Dn,-(An)             Dn -(An) l. bchg,
bchg.l Dn,(An)+             Dn (An)+ l. bchg,
bchg.l Dn,d(An)             Dn n d(An) l. bchg,
bchg.l Dn,d(An,Dn.l)        Dn n d(An, Dn.l) l. bchg,
bchg.l Dn,$d                Dn $ n l. bchg,
bchg.l Dn,$var              Dn $ var l. bchg,

bchg.b #i,Dn                # n Dn b. bchg,
bchg.b #i,(An)              # n (An) b. bchg,
bchg.b #i,-(An)             # n -(An) b. bchg,
bchg.b #i,(An)+             # n (An)+ b. bchg,
bchg.b #i,d(An)             # n n d(An) b. bchg,
bchg.b #i,d(An,Dn.l)        # n n d(An, Dn.l) b. bchg,
```

```
bchg.b #i,$d                # n $ n b. bchg,
bchg.b #i,$var              # n $ var b. bchg,
bchg.l #i,Dn                # n Dn l. bchg,
bchg.l #i,(An)              # n (An) l. bchg,
bchg.l #i,-(An)             # n -(An) l. bchg,
bchg.l #i,(An)+             # n (An)+ l. bchg,
bchg.l #i,d(An)             # n n d(An) l. bchg,
bchg.l #i,d(An,Dn.l)        # n n d(An, Dn.l) l. bchg,
bchg.l #i,$d                # n $ n l. bchg,
bchg.l #i,$var              # n $ var l. bchg,

bclr.b Dn,Dn                Dn Dn b. bclr,
bclr.b Dn,(An)              Dn (An) b. bclr,
bclr.b Dn,-(An)             Dn -(An) b. bclr,
bclr.b Dn,(An)+             Dn (An)+ b. bclr,
bclr.b Dn,d(An)             Dn n d(An) b. bclr,
bclr.b Dn,d(An,Dn.l)        Dn n d(An, Dn.l) b. bclr,
bclr.b Dn,$d                Dn $ n b. bclr,
bclr.b Dn,$var              Dn $ var b. bclr,
bclr.l Dn,Dn                Dn Dn l. bclr,
bclr.l Dn,(An)              Dn (An) l. bclr,
bclr.l Dn,-(An)             Dn -(An) l. bclr,
bclr.l Dn,(An)+             Dn (An)+ l. bclr,
bclr.l Dn,d(An)             Dn n d(An) l. bclr,
bclr.l Dn,d(An,Dn.l)        Dn n d(An, Dn.l) l. bclr,
bclr.l Dn,$d                Dn $ n l. bclr,
bclr.l Dn,$var              Dn $ var l. bclr,

bclr.b #i,Dn                # n Dn b.bclr,
bclr.b #i,(An)              # n (An) b. bclr,
bclr.b #i,-(An)             # n -(An ) b. bclr,
bclr.b #i,(An)+             # n (An)+ b. bclr,
bclr.b #i,d(An)             # n n d(An) b. bclr,
bclr.b #i,d(An,Dn.l)        # n n d(An, Dn.l) b. bclr,
bclr.b #i,$d                # n $ n b. bclr,
bclr.b #i,$var              # n $ var b. bclr,
bclr.l #i,Dn                # n Dn l. bclr,
bclr.l #i,(An)              # n (An) l. bclr,
bclr.l #i,-(An)             # n -(An) l. bclr,
bclr.l #i,(An)+             # n (An)+ l. bclr,
bclr.l #i,d(An)             # n n d(An) l. bclr,
bclr.l #i,d(An,Dn.l)        # n n d(An, Dn.l) l. bclr,
bclr.l #i,$d                # n $ n l. bclr,
bclr.l #i,$var              # n $ var l. bclr,

bgnd                        bgnd,

bkpt #i                     # n bkpt,
```

```
bra label                   label bra,

bset.b Dn,Dn                Dn Dn b. bset,
bset.b Dn,(An)              Dn (An) b. bset,
bset.b Dn,-(An)             Dn -(An) b. bset,
bset.b Dn,(An)+             Dn (An)+ b. bset,
bset.b Dn,d(An)             Dn n d(An) b. bset,
bset.b Dn,d(An,Dn.l)        Dn n d(An, Dn.l) b. bset,
bset.b Dn,$d                Dn $ n b. bset,
bset.b Dn,$var              Dn $ var b. bset,
bset.l Dn,Dn                Dn Dn l. bset,
bset.l Dn,(An)              Dn (An) l. bset,
bset.l Dn,-(An)             Dn -(An) l. bset,
bset.l Dn,(An)+             Dn (An)+ l. bset,
bset.l Dn,d(An)             Dn n d(An) l. bset,
bset.l Dn,d(An,Dn.l)        Dn n d(An, Dn.l) l. bset,
bset.l Dn,$d                Dn $ n l. bset,
bset.l Dn,$var              Dn $ var l. bset,

bset.b #i,Dn                # n Dn b. bset,
bset.b #i,(An)              # n (An) b. bset,
bset.b #i,-(An)             # n -(An) b. bset,
bset.b #i,(An)+             # n (An)+ b. bset,
bset.b #i,d(An)             # n n d(An) b. bset,
bset.b #i,d(An,Dn.l)        # n n d(An, Dn.l) b. bset,
bset.b #i,$d                # n $ n b. bset,
bset.b #i,$var              # n $ var b. bset,
bset.l #i,Dn                # n Dn l. bset,
bset.l #i,(An)              # n (An) l. bset,
bset.l #i,-(An)             # n -(An) l. bset,
bset.l #i,(An)+             # n (An)+ l. bset,
bset.l #i,d(An)             # n n d(An) l. bset,
bset.l #i,d(An,Dn.l)        # n n d(An, Dn.l) l. bset,
bset.l #i,$d                # n $ n l. bset,
bset.l #i,$var              # n $ var l. bset,

bsr label                   label bsr,

btst.b Dn,Dn                Dn Dn b. btst,
btst.b Dn,(An)              Dn (An) b. btst,
btst.b Dn,-(An)             Dn -(An) b. btst,
btst.b Dn,(An)+             Dn (An)+ b. btst,
btst.b Dn,d(An)             Dn n d(An) b. btst,
btst.b Dn,d(An,Dn.l)        Dn n d(An, Dn.l) b. btst,
btst.b Dn,$d                Dn $ n b. btst,
btst.b Dn,$var              Dn $ var b. btst,
btst.l Dn,Dn                Dn Dn l. btst,
```

```
btst.l Dn,(An)              Dn (An) l. btst,
btst.l Dn,-(An)             Dn -(An) l. btst,
btst.l Dn,(An)+             Dn (An)+ l. btst,
btst.l Dn,d(An)             Dn n d(An) l. btst,
btst.l Dn,d(An,Dn.l)        Dn n d(An, Dn.l) l. btst,
btst.l Dn,$d                Dn $ n l. btst,
btst.l Dn,$var              Dn $ var l. btst,

btst.b #i,Dn                # n Dn b. btst,
btst.b #i,(An)              # n (An) b. btst,
btst.b #i,-(An)             # n -(An) b. btst,
btst.b #i,(An)+             # n (An)+ b. btst,
btst.b #i,d(An)             # n n d(An) b. btst,
btst.b #i,d(An,Dn.l)        # n n d(An, Dn.l) b. btst,
btst.b #i,$d                # n $ n b. btst,
btst.b #i,$var              # n $ var b. btst,
btst.l #i,Dn                # n Dn l. btst,
btst.l #i,(An)              # n (An) l. btst,
btst.l #i,-(An)             # n -(An) l. btst,
btst.l #i, (An)+            # n (An)+ l. btst,
btst.l #i,d(An)             # n n d(An) l. btst,
btst.l #i,Dn.l)             # n n d(An, Dn.l) l. btst,
btst.l #i,$d                # n $ n l. btst,
btst.l #i,$var              # n $ var l. btst,

chk.l Dn,Dn                 Dn Dn l. chk,
chk.l (An),Dn               (An) Dn l. chk,
chk.l -(An),Dn              -(An) Dn l. chk,
chk.l (An)+,Dn              (An)+ Dn l. chk,
chk.l d(An),Dn              n d(An) Dn l. chk,
chk.l d(An,Dn.l),Dn         n d(An, Dn.l) Dn l. chk,
chk.l $d,Dn                 $ n Dn l. chk,
chk.l $var,Dn               $ var Dn l. chk,
chk.l #i,Dn                 # n Dn l. chk,

chk2.b (An),Dn              (An) Dn b. chk2,
chk2.b d(An),Dn             n d(An) Dn b. chk2,
chk2.b d(A n,Dn.l),Dn       n d(An, Dn.l) Dn b. chk2,
chk2.b $d,Dn                $ n Dn b. chk2,
chk2.b $var,Dn              $ var Dn b. chk2,
chk2.w (An),Dn              (An) Dn w. chk2,
chk2.w d(An),Dn             n d(An) Dn w. chk2,
chk2.w d(An,Dn.l),Dn        n d(An, Dn.l) Dn w. chk2,
chk2.w $d,Dn                $ n Dn w. chk2,
chk2.w $var,Dn              $ var Dn w. chk2,
chk2.l (An),Dn              (An) Dn l. chk2,
chk2.l d(An) Dn             n d(An) Dn l. chk2,
chk2.l d(An,Dn.l),Dn        n d(An, Dn.l) Dn l. chk2,
```

```
chk2.l $d,Dn                $ n Dn l. chk2,
chk2.l $var,Dn              $ var Dn l. chk2,

clr.b Dn                    Dn b. clr,
clr.b (An)                  (An) b. clr,
clr.b -(An)                 -(An) b. clr,
clr.b (An)+                 (An)+ b. clr,
clr.b d(An)                 n d(An) b. clr,
clr.b d(An,Dn.l)            n d(An, Dn.l) b. clr,
clr.b $d                    $ n b. clr,
clr.b $var                  $ var b. clr,
clr.l Dn                    Dn l. clr,
clr.l (An)                  (An) l. clr,
clr.l -(An)                 -(An) l. clr,
clr.l (An)+                 (An)+ l. clr,
clr.l d(An)                 n d(An) l. clr,
clr.l d(An,Dn.l)            n d(An, Dn.l) l. clr,
clr.l $d                    $ n l. clr,
clr.l $var                  $ var l. clr,

cmp.b Dn,Dn                 Dn Dn b. cmp,
cmp.b (An),Dn               (An) Dn b. cmp,
cmp.b -(An),Dn              -(An) Dn b. cmp,
cmp.b (An)+,Dn              (An)+ Dn b. cmp,
cmp.b d(An),Dn              n d(An) Dn b. cmp,
cmp.b d(An,Dn.l),Dn         n d(An, Dn.l) Dn b. cmp,
cmp.b $d,Dn                 $ n Dn b. cmp,
cmp.b $var,Dn               $ var Dn b. cmp,
cmp.b #i,Dn                 # n Dn b. cmp,
cmp.w Dn,Dn                 Dn Dn w. cmp,
cmp.w (An),Dn               (An) Dn w. cmp,
cmp.w -(An),Dn              -(An) Dn w. cmp,
cmp.w (An)+,Dn              (An)+ Dn w. cmp,
cmp.w d(An),Dn              n d(An) Dn w. cmp,
cmp.w d(An,Dn.l),Dn         n d(An, Dn.l) Dn w. cmp,
cmp.w $d,Dn                 $ n Dn w. cmp,
cmp.w $var,Dn               $ var Dn w. cmp,
cmp.w #i Dn                 # n Dn w. cmp,
cmp.l Dn,Dn                 Dn Dn l. cmp,
cmp.l (An),Dn               (An) Dn l. cmp,
cmp.l -(An),Dn              -(An) Dn l. cmp,
cmp.l (An)+,Dn              (An)+ Dn l. cmp,
cmp.l d(An),Dn              n d(An) Dn l. cmp,
cmp.l d(An,Dn.l),Dn         n d(An, Dn.l) Dn l. cmp,
cmp.l $d,Dn                 $ n Dn l. cmp,
cmp.l $var,Dn               $ var Dn l. cmp,
cmp.l #i,Dn                 # n Dn l. cmp,
```

```
cmp2.b (An),Dn              (An) Dn b. cmp2,
cmp2.b d(An),Dn             n d(An) Dn b. cmp2,
cmp2.b d(An,Dn.l),Dn        n d(An, Dn.l) Dn b. cmp2,
cmp2.b $d,Dn                $ n Dn b. cmp2,
cmp2.b $var,Dn              $ var Dn b. cmp2,
cmp2.w (An),Dn              (An) Dn w. cmp2,
cmp2.w d(An),Dn             n d(An) Dn w. cmp2,
cmp2.w d(An,Dn.l),Dn        n d(An, Dn.l) Dn w. cmp2,
cmp2.w $d,Dn                $ n Dn w. cmp2,
cmp2.w $var,Dn              $ var Dn w. cmp2,
cmp2.l (An),Dn              (An) Dn l. cmp2,
cmp2.l d(An),Dn             n d(An) Dn l. cmp2,
cmp2.l d(An,Dn.l),Dn        n d(An, Dn.l) Dn l. cmp2,
cmp2.l $d,Dn                $ n Dn l. cmp2,
cmp2.l $var,Dn              $ var Dn l. cmp2,

cmpa.w An,An                An An w. cmpa,
cmpa.w (An),An              (An) An w. cmpa,
cmpa.w -(An),An             -(An) An w. cmpa,
cmpa.w (An)+,An             (An)+ An w. cmpa,
cmpa.w d(An),An             n d(An) An w. cmpa,
cmpa.w d(An,Dn.l),An        n d(An, Dn.l) An w. cmpa,
cmpa.w $d,An                $ n An w. cmpa,
cmpa.w $var,An              $ var An w. cmpa,
cmpa.w #i,An                # n An w. cmpa,
cmpa.l An,An                An An l. cmpa,
cmpa.l (An),An              (An) An l. cmpa,
cmpa.l -(An),An             -(An) An l. cmpa,
cmpa.l (An)+,An             (An)+ An l. cmpa,
cmpa.l d(An),An             n d(An) An l. cmpa,
cmpa.l d(An,Dn.l),An        n d(An, Dn.l) An l. cmpa,
cmpa.l $d,An                $ n An l. cmpa,
cmpa.l $var,An              $ var An l. cmpa,
cmpa.l #i,An                # n An l. cmpa,

cmpi.b #i,Dn                # n Dn b. cmpi,
cmpi.b #i,(An)              # n (An) b. cmpi,
cmpi.b #i,-(An)             # n -(An) b. cmpi,
cmpi.b #i,(An)+             # n (An)+ b. cmpi,
cmpi.b #i,d(An)             # n n d(An) b. cmpi,
cmpi.b #i,d(An,Dn.l)        # n n d(An, Dn.l) b. cmpi,
cmpi.b #i,$d                # n $ n b. cmpi,
cmpi.b #i,$var              # n $ var b. cmpi,
cmpi.w #i,Dn                # n Dn w. cmpi,
cmpi.w #i,(An)              # n (An) w. cmpi,
cmpi.w #i,-(An)             # n -(An) w. cmpi,
cmpi.w #i,(An)+             # n (An)+ w. cmpi,
cmpi.w #i,d(An)             # n n d(An) w. cmpi,
```

```
cmpi.w #i,d(An,Dn.l)        # n n d(An, Dn.l) w. cmpi,
cmpi.w #i,$d                # n $ n w. cmpi,
cmpi.w #i,$var              # n $ var w. cmpi,
cmpi.l #i,Dn                # n Dn l. cmpi,
cmpi.l #i,(An)              # n (An) l. cmpi,
cmpi.l #i,-(An)             # n -(An) l. cmpi,
cmpi.l #i,(An)+             # n (An)+ l. cmpi,
cmpi.l #i,d(An)             # n n d(An) l. cmpi,
cmpi.l #i,d(An,Dn.l)        # n n d(An, Dn.l) l. cmpi,
cmpi.l #i,$d                # n $ n l. cmpi,
cmpi.l #i,$var              # n $ var l. cmpi,

cmpm.b (An)+,(An)+          (An)+ (An)+ b. cmpm,
cmpm.w (An)+,(An)+          (An)+ (An)+ w. cmpm,
cmpm.l (An)+,(An)+          (An)+ (An)+ l. cmpm,

dbhi Dn,label               Dn label dbhi,
dbls Dn,label               Dn label dbls,
dbcc Dn,label               Dn label dbcc,
dbcs Dn,label               Dn label dbcs,
dbne Dn,label               Dn label dbne,
dbeq Dn,label               Dn label dbeq,
dbvc Dn,label               Dn label dbvc,
dbvs Dn,label               Dn label dbvs,
dbpl Dn,label               Dn label dbpl,
dbmi Dn,label               Dn label dbmi,
dbge Dn,label               Dn label dbge,
dblt Dn,label               Dn label dblt,
dbgt Dn,label               Dn label dbgt,
dble Dn,label               Dn label dble,

dbra Dn,label               Dn label dbra,

divs.w Dn,Dn                Dn Dn w. divs,
divs.w (An),Dn              (An) Dn w. divs,
divs.w -(An),Dn             -(An) Dn w. divs,
divs.w (An)+,Dn             (An)+ Dn w. divs,
divs.w d(An),Dn             n d(An) Dn w. divs,
divs.w d(An,Dn.l),Dn        n d(An, Dn.l) Dn w. divs,
divs.w $d,Dn                $ n Dn w. divs,
divs.w $var,Dn              $ var Dn w. divs,
divs.w #i,Dn                # n Dn w. divs,

divs.l Dn,Dn                Dn Dn l. divs,
divs.l (An),Dn              (An) Dn l. divs,
divs.l -(An),Dn             -(An) Dn l. divs,
divs.l (An)+,Dn             (An)+ Dn l. divs,
divs.l d(An),Dn             n d(An) Dn l. divs,
```

```
divs.l d(An,Dn.l),Dn        n d(An, Dn.l) Dn l.divs,
divs.l $d,Dn                $ n Dn l. divs,
divs.l $var,Dn              $ var Dn l. divs,
divs.l #i,Dn                # n Dn l. divs,

divs.l Dn,Dn:Dn             Dn Dn: Dn l. divs,
divs.l (An),Dn:Dn           (An) Dn: Dn l. divs,
divs.l -(An),Dn:Dn          -(An) Dn: Dn l. divs,
divs.l (An)+,Dn:Dn          (An)+ Dn: Dn l. divs,
divs.l d(An),Dn:Dn          n d(An) Dn: Dn l. divs,
divs.l d(An,Dn.l),Dn:Dn     n d(An, Dn.l) Dn: Dn l. divs,
divs.l $d,Dn:Dn             $ n Dn: Dn l. divs,
divs.l $var,Dn:Dn           $ var Dn: Dn l. divs,
divs.l #i,Dn:Dn             # n Dn: Dn l.divs,

divsl.l Dn,Dn:Dn            Dn Dn: Dn l. divsl,
divsl.l (An),Dn:Dn          (An) Dn: Dn l. divsl,
divsl.l -(An),Dn:Dn         -(An) Dn: Dn l. divsl,
divsl.l (An)+,Dn:Dn         (An)+ Dn: Dn l. divsl,
divsl.l d(An),Dn:Dn         n d(An) Dn: Dn l. divsl,
divsl.l d(An,Dn.l),Dn:Dn    n d(An, Dn.l) Dn: Dn l. divsl,
divsl.l $d,Dn:Dn            $ n Dn: Dn l. divsl,
divsl.l $var,Dn:Dn          $ var Dn: Dn l. divsl,
divsl.l #i,Dn:Dn            # n Dn: Dn l. divsl,

divu.w Dn,Dn                Dn Dn w. divu,
divu.w (An),Dn              (An) Dn w. divu,
divu.w -(An),Dn             -(An) Dn w. divu,
divu.w (An)+,Dn             (An)+ Dn w. divu,
divu.w d(An),Dn             n d(An) Dn w. divu,
divu.w d(An,Dn.l),Dn        n d(An, Dn.l) Dn w. divu,
divu.w $d,Dn                $ n Dn w. divu,
divu.w $var,Dn              $ var Dn w. divu,
divu.w #i,Dn                # n Dn w. divu,

divu.l Dn,Dn                Dn Dn l. divu,
divu.l (An),Dn              (An) Dn l. divu,
divu.l -(An),Dn             -(An) Dn l. divu,
divu.l (An)+,Dn             (An)+ Dn l. divu,
divu.l d(An),Dn             n d(An) Dn l. divu,
divu.l d(An,Dn.l),Dn        n d(An, Dn.l) Dn l. divu,
divu.l $d,Dn                $ n Dn l. divu,
divu.l $var,Dn              $ var Dn l. divu,
divu.l #i,Dn                # n Dn l. divu,

divu.l Dn,Dn:Dn             Dn Dn: Dn l. divu,
divu.l (An),Dn:Dn           (An) Dn: Dn l. divu,
divu.l -(An),Dn:Dn          -(An) Dn: Dn l. divu,
```

```
divu.l (An)+,Dn:Dn          (An)+ Dn: Dn l. divu,
divu.l d(An),Dn:Dn          n d(An) Dn: Dn l. divu,
divu.l d(An,Dn.l),Dn:Dn     n d(An, Dn.l) Dn: Dn l. divu,
divu.l $d,Dn:Dn             $ n Dn: Dn l. divu,
divu.l $var,Dn:Dn           $ var Dn: Dn l. divu,
divu.l #i,Dn:Dn             # n Dn: Dn l. divu,

divul.l Dn,Dn:Dn            Dn Dn: Dn l. divul,
divul.l (An),Dn:Dn          (An) Dn: Dn l. divul,
divul.l -(An),Dn:Dn         -(An) Dn: Dn l. divul,
divul.l (An)+,Dn:Dn         (An)+ Dn: Dn l. divul,
divul.l d(An),Dn:Dn         n d(An) Dn: Dn l. divul,
divul.l d(An,Dn.l),Dn:Dn    n d(An, Dn.l) Dn: Dn l. divul,
divul.l $d,Dn:Dn            $ n Dn: Dn l. divul,
divul.l $var,Dn:Dn          $ var Dn: Dn l. divul,
divul.l #i,Dn:Dn            # n Dn: Dn l. divul,

eor.b Dn,Dn                 Dn Dn b. eor,
eor.b Dn,(An)               Dn (An) b. eor,
eor.b Dn,-(An)              Dn -(An) b. eor,
eor.b Dn,(An)+              Dn (An)+ b. eor,
eor.b Dn,d(An)              Dn n d(An) b. eor,
eor.b Dn,d(An,Dn.l)         Dn n d(An, Dn.l) b. eor,
eor.b Dn,$d                 Dn $ n b. eor,
eor.b Dn,$var               Dn $ var b. eor,
eor.l Dn,Dn                 Dn Dn l. eor,
eor.l Dn,(An)               Dn (An) l. eor,
eor.l Dn,-(An)              Dn -(An) l. eor,
eor.l Dn,(An)+              Dn (An)+ l. eor,
eor.l Dn,d(An)              Dn n d(An) l. eor,
eor.l Dn,d(An,Dn.l)         Dn n d(An, Dn.l) l. eor,
eor.l Dn,$d                 Dn $ n l. eor,
eor.l Dn,$var               Dn $ var l. eor,

eori.b #i,Dn                # n Dn b. eori,
eori.b #i,(An)              # n (An) b. eori,
eori.b #i,-(An)             # n -(An) b. eori,
eori.b #i,(An)+             # n (An)+ b. eori,
eori.b #i,d(An)             # n n d(An) b. eori,
eori.b #i,d(An,Dn.l)        # n n d(An, Dn.l) b. eori,
eori.b #i,$d                # n $ n b. eori,
eori.b #i,$var              # n $ var b. eori,
eori.l #i,Dn                # n Dn l. eori,
eori.l #i,(An)              # n (An) l. eori,
eori.l #i,-(An)             # n -(An) l. eori,
eori.l #i,(An)+             # n (An)+ l. eori,
eori.l #i,d(An)             # n n d(An) l. eori,
eori.l #i,d(An,Dn.l)        # n n d(An, Dn.l) l. eori,
```

```
eori.l #i,$d                # n $ n l. eori,
eori.l #i,$var              # n $ var l. eori,

eori.b #i,ccr               # n ccr b. eori,

eori.w #i,sr                # n sr w. eori,

exg.l Dn,Dn                 Dn Dn l. exg,
exg.l An,An                 An An l. exg,
exg.l Dn,An                 Dn An l. exg,
exg.l An,Dn                 An Dn l. exg,

ext.w Dn                    Dn w. ext,
ext.l Dn                    Dn l. ext,

extb.l Dn                   Dn l. extb,

jmp (An)                    (An) jmp,
jmp d(An)                   n d(An) jmp,
jmp d(An,Dn.l)              n d(An, Dn.l) jmp,
jmp $d                      $ n jmp,
jmp $var                    $ var jmp,

jsr (An)                    (An) jsr,
jsr d(An)                   n d(An) jsr,
jsr d(An,Dn.l)              n d(An, Dn.l) jsr,
jsr $d                      $ n jsr,
jsr $var                    $ var jsr,

lea (An),An                 (An) An lea,
lea d(An),An                n d(An) An lea,
lea d(An,Dn.l),An           n d(An, Dn.l) An lea,
lea $d,An                   $ n An lea,
lea $var,An                 $ var An lea,

link An, #i                 # -n An link,

lpstop #i                   # n lpstop,

lsl.b Dn,Dn                 Dn Dn b. lsl,
lsl.b #i,Dn                 # n Dn b. lsl,
lsl.w Dn,Dn                 Dn Dn w. lsl,
lsl.w #i,Dn                 # n Dn w. lsl,
lsl.l Dn,Dn                 Dn Dn l. lsl,
lsl.l #i,Dn                 # n Dn l. lsl,

lsl.b Dn                    # 1 Dn b. lsl,
lsl.b (An)                  # 1 (An) b. lsl,
```

```
lsl.b -(An)                 # 1 -(An) b. lsl,
lsl.b (An)+                 # 1 (An)+ b. lsl,
lsl.b d(An)                 # 1 n d(An) b. lsl,
lsl.b d(An,Dn.l)            # 1 n d(An, Dn.l) b. lsl,
lsl.b $d                    # 1 $ n b. lsl,
lsl.b $var                  # 1 $ var b. lsl,
lsl.w Dn                    # 1 Dn w. lsl,
lsl.w (An)                  # 1 (An) w. lsl,
lsl.w -(An)                 # 1 -(An) w. lsl,
lsl.w (An)+                 # 1 (An)+ w. lsl,
lsl.w d(An)                 # 1 n d(An) w. lsl,
lsl.w d(An,Dn.l)            # 1 n d(An, Dn.l) w. lsl,
lsl.w $d                    # 1 $ n w. lsl,
lsl.w $var                  # 1 $ var w. lsl,
lsl.l Dn                    # 1 Dn l. lsl,
lsl.l (An)                  # 1 (An) l. lsl,
lsl.l -(An)                 # 1 -(An) l. lsl,
lsl.l (An)+                 # 1 (An)+ l. lsl,
lsl.l d(An)                 # 1 n d(An) l. lsl,
lsl.l d(An,Dn.l)            # 1 n d(An, Dn.l) l. lsl,
lsl.l $d                    # 1 $ n l. lsl,
lsl.l $var                  # 1 $ var l. lsl,

lsr.b Dn,Dn                 Dn Dn b. lsr,
lsr.b #i,Dn                 # n Dn b. lsr,
lsr.w Dn,Dn                 Dn Dn w. lsr,
lsr.w #i,Dn                 # n Dn w. lsr,
lsr.l Dn,Dn                 Dn Dn l. lsr,
lsr.l #i,Dn                 # n Dn l. lsr,

lsr.b Dn                    # 1 Dn b.lsr,
lsr.b (An)                  # 1 (An) b. lsr,
lsr.b -(An)                 # 1 -(An) b. lsr,
lsr.b (An)+                 # 1 (An)+ b. lsr,
lsr.b d(An)                 # 1 n d(An) b. lsr,
lsr.b d(An,Dn.l)            # 1 n d(An, Dn.l) b. lsr,
lsr.b $d                    # 1 $ n b. lsr,
lsr.b $var                  # 1 $ var b. lsr,
lsr.w Dn                    # 1 Dn w. lsr,
lsr.w (An)                  # 1 (An) w. lsr,
lsr.w -(An)                 # 1 -(An) w. lsr,
lsr.w (An)+                 # 1 (An)+ w. lsr,
lsr.w d(An)                 # 1 n d(An) w. lsr,
lsr.w d(An,Dn.l)            # 1 n d(An, Dn.l) w. lsr,
lsr.w $d                    # 1 $ n w. lsr,
lsr.w $var                  # 1 $ var w. lsr,
lsr.l Dn                    # 1 Dn l. lsr,
lsr.l (An)                  # 1 (An) l. lsr,
```

```
lsr.l -(An)                 # 1 -(An) l. lsr,
lsr.l (An)+                 # 1 (An)+ l. lsr,
lsr.l d(An)                 # 1 n d(An) l. lsr,
lsr.l d(An,Dn.l)            # 1 n d(An, Dn.l) l. lsr,
lsr.l $d                    # 1 $ n l. lsr,
lsr.l $var                  # 1 $ var l. lsr,

move.b Dn,Dn                Dn Dn b. move,
move.b An,An                An An b. move,
move.b (An),Dn              (An) Dn b. move,
move.b -(An),Dn             -(An) Dn b. move,
move.b (An)+,Dn             (An)+ Dn b. move,
move.b d(An),Dn             n d(An) Dn b. move,
move.b d(An,Dn.l),Dn        n d(An, Dn.l) Dn b. move,
move.b $d,Dn                $ n Dn b. move,
move.b $var,Dn              $ var Dn b. move,
move.w Dn,Dn                Dn Dn w. move,
move.w An,An                An An w. move,
move.w (An),Dn              (An) Dn w. move,
move.w -(An) Dn             -(An) Dn w. move,
move.w (An)+ Dn             (An)+ Dn w. move,
move.w d(An) Dn             n d(An) Dn w. move,
move.w d(An,Dn.l) Dn        n d(An, Dn.l) Dn w. move,
move.w $d Dn                $ n Dn w. move,
move.w $var Dn              $ var Dn w. move,
move.l Dn,Dn                Dn Dn l. move,
move.l An,An                An An l. move,
move.l (An),Dn              (An) Dn l. move,
move.l -(An),Dn             -(An) Dn l. move,
move.l (An)+,Dn             (An)+ Dn l. move,
move.l d(An),Dn             n d(An) Dn l. move,
move.l d(An,Dn.l),Dn        n d(An, Dn.l) Dn l. move,
move.l $d,Dn                $ n Dn l. move,
move.l $var,Dn              $ var Dn l. move,

move.b Dn,Dn                Dn Dn b. move,
move.b An,An                An An b. move,
move.b Dn,(An)              Dn (An) b. move,
move.b Dn,-(An)             Dn -(An) b. move,
move.b Dn,(An)+             Dn (An)+ b. move,
move.b Dn,d(An)             Dn n d(An) b. move,
move.b Dn,d(An,Dn.l)        Dn n d(An, Dn.l) b. move,
move.b Dn,$d                Dn $ n b. move,
move.b Dn,$var              Dn $ var b. move,
move.w Dn,Dn                Dn Dn w. move,
move.w An,An                An An w. move,
move.w Dn,(An)              Dn (An) w. move,
move.w Dn,-(An)             Dn -(An) w. move,
```

```
move.w Dn,(An)+             Dn (An)+ w. move,
move.w Dn,d(An)             Dn n d(An) w. move,
move.w Dn,d(An,Dn.l)        Dn n d(An, Dn.l) w. move,
move.w Dn,$d                Dn $ n w. move,
move.w Dn,$var              Dn $ var w. move,
move.l Dn,Dn                Dn Dn l. move,
move.l An,An                An An l. move,
move.l Dn,(An)              Dn (An) l. move,
move.l Dn,-(An)             Dn -(An) l. move,
move.l Dn,(An)+             Dn (An)+ l. move,
move.l Dn,d(An)             Dn n d(An) l. move,
move.l Dn,d(An,Dn.l)        Dn n d(An, Dn.l) l. move,
move.l Dn,$d                Dn $ n l. move,
move.l Dn,$var              Dn $ var l. move,

move.b Dn,An                Dn An b. move,
move.b An,Dn                An Dn b. move,
move.w Dn,An                Dn An w. move,
move.w An,Dn                An Dn w. move,
move.l Dn,An                Dn An l. move,
move.l An,Dn                An Dn l. move,

move.b An,(An)              An (An) b. move,
move.w An,(An)              An (An) w. move,
move.l An,(An)              An (An) l. move,

move.b An,-(An)             An -(An) b. move,
move.w An,-(An)             An -(An) w. move,
move.l An,-(An)             An -(An) l. move,

move.b An,(An)+             An (An)+ b. move,
move.w An,(An)+             An (An)+ w. move,
move.l An,(An)+             An (An)+ l. move,

move.b An,d(An)             An n d(An) b. move,
move.w An,d(An)             An n d(An) w. move,
move.l An,d(An)             An n d(An) l. move,

move.b An,d(An,Dn.l)        An n d(An, Dn.l) b. move,
move.w An,d(An,Dn.l)        An n d(An, Dn.l) w. move,
move.l An,d(An,Dn.l)        An n d(An, Dn.l) l. move,

move.b An,$d                An $ n b. move,
move.w An,$d                An $ n w. move,
move.l An,$d                An $ n l. move,

move.b (An),An              (An) An b. move,
move.w (An),An              (An) An w. move,
```

```
move.l (An),An              (An) An l. move,

move.b (An),(An)            (An) (An) b. move,
move.w (An),(An)            (An) (An) w. move,
move.l (An),(An)            (An) (An) l. move,

move.b (An),-(An)           (An) -(An) b. move,
move.w (An),-(An)           (An) -(An) w. move,
move.l (An),-(An)           (An) -(An) l. move,

move.b (An),(An)+           (An) (An)+ b. move,
move.w (An),(An)+           (An) (An)+ w. move,
move.l (An),(An)+           (An) (An)+ l. move,

move.b (An),d(An)           (An) n d(An) b. move,
move.w (An),d(An)           (An) n d(An) w. move,
move.l (An),d(An)           (An) n d(An) l. move,

move.b (An),d(An,Dn.l)      (An) n d(An, Dn.l) b. move,
move.w (An),d(An,Dn.l)      (An) n d(An, Dn.l) w. move,
move.l (An),d(An,Dn.l)      (An) n d(An, Dn.l) l. move,
move.b (An),$d              (An) $ n b. move,
move.w (An),$d              (An) $ n w. move,
move.l (An),$d              (An) $ n l. move,

move.b -(An),An             -(An) An b. move,
move.w -(An),An             -(An) An w. move,
move.l -(An),An             -(An) An l. move,

move.b -(An),(An)           -(An) (An) b. move,
move.w -(An),(An)           -(An) (An) w. move,
move.l -(An),(An)           -(An) (An) l. move,

move.b -(An),-(An)          -(An) -(An) b. move,
move.w -(An),-(An)          -(An) -(An) w. move,
move.l -(An),-(An)          -(An) -(An) l. move,

move.b -(An),(An)+          -(An) (An)+ b. move,
move.w -(An),(An)+          -(An) (An)+ w. move,
move.l -(An),(An)+          -(An) (An)+ l. move,

move.b -(An),d(An)          -(An) n d(An) b. move,
move.w -(An),d(An)          -(An) n d(An) w. move,
move.l -(An),d(An)          -(An) n d(An) l. move,

move.b -(An),d(An,Dn.l)     -(An) n d(An, Dn.l) b. move,
move.w -(An),d(An,Dn.l)     -(An) n d(An, Dn.l) w. move,
move.l -(An),d(An,Dn.l)     -(An) n d(An, Dn.l) l. move,
```

```
move.b -(An),$d             -(An) $ n b. move,
move.w -(An),$d             -(An) $ n w. move,
move.l -(An),$d             -(An) $ n l. move,

move.b (An)+,An             (An)+ An b. move,
move.w (An)+,An             (An)+ An w. move,
move.l (An)+,An             (An)+ An l. move,

move.b (An)+,(An)           (An)+ (An) b. move,
move.w (An)+,(An)           (An)+ (An) w. move,
move.l (An)+,(An)           (An)+ (An) l. move,

move.b (An)+,-(An)          (An)+ -(An) b. move,
move.w (An)+,-(An)          (An)+ -(An) w. move,
move.l (An)+,-(An)          (An)+ -(An) l. move,

move.b (An)+,(An)+          (An)+ (An)+ b. move,
move.w (An)+,(An)+          (An)+ (An)+ w. move,
move.l (An)+,(An)+          (An)+ (An)+ l. move,

move.b (An)+,d(An)          (An)+ n d(An) b. move,
move.w (An)+,d(An)          (An)+ n d(An) w. move,
move.l (An)+,d(An)          (An)+ n d(An) l. move,

move.b (An)+,d(An,Dn.l)     (An)+ n d(An, Dn.l) b. move,
move.w (An)+,d(An,Dn.l)     (An)+ n d(An, Dn.l) w. move,
move.l (An)+,d(An,Dn.l)     (An)+ n d(An, Dn.l) l. move,

move.b (An)+,$d             (An)+ $ n b. move,
move.w (An)+,$d             (An)+ $ n w. move,
move.l (An)+,$d             (An)+ $ n l. move,

move.b d(An),An             n d(An) An b. move,
move.w d(An),An             n d(An) An w. move,
move.l d(An),An             n d(An) An l. move,

move.b d(An),(An)           n d(An) (An) b. move,
move.w d(An),(An)           n d(An) (An) w. move,
move.l d(An),(An)           n d(An) (An) l. move,

move.b d(An),-(An)          n d(An) -(An) b. move,
move.w d(An),-(An)          n d(An) -(An) w. move,
move.l d(An),-(An)          n d(An) -(An) l. move,

move.b d(An),(An)+          n d(An) (An)+ b. move,
move.w d(An),(An)+          n d(An) (An)+ w. move,
move.l d(An),(An)+          n d(An) (An)+ l. move,
```

```
move.b d(An),d(An)          n d(An) n d(An) b. move,
move.w d(An),d(An)          n d(An) n d(An) w. move,
move.l d(An),d(An)          n d(An) n d(An) l. move,

move.b d(An),d(An,Dn.l)     n d(An) n d(An, Dn.l) b. move,
move.w d(An),d(An,Dn.l)     n d(An) n d(An, Dn.l) w. move,
move.l d(An),d(An,Dn.l)     n d(An) n d(An, Dn.l) l. move,

move.b d(An),$d             n d(An) $ n b. move,
move.w d(An),$d             n d(An) $ n w. move,
move.l d(An),$d             n d(An) $ n l. move,

move.b d(An,Dn.l),An        n d(An, Dn.l) An b. move,
move.w d(An,Dn.l),An        n d(An, Dn.l) An w. move,
move.l d(An,Dn.l),An        n d(An, Dn.l) An l. move,

move.b d(An,Dn.l),(An)      n d(An, Dn.l) (An) b. move,
move.w d(An,Dn.l),(An)      n d(An, Dn.l) (An) w. move,
move.l d(An,Dn.l),(An)      n d(An, Dn.l) (An) l. move,

move.b d(An,Dn.l),-(An)     n d(An, Dn.l) -(An) b. move,
move.w d(An,Dn.l),-(An)     n d(An, Dn.l) -(An) w. move,
move.l d(An,Dn.l),-(An)     n d(An, Dn.l) -(An) l. move,

move.b d(An,Dn.l),(An)+     n d(An, Dn.l) (An)+ b. move,
move.w d(An,Dn.l),(An)+     n d(An, Dn.l) (An)+ w. move,
move.l d(An,Dn.l),(An)+     n d(An, Dn.l) (An)+ l. move,

move.b d(An,Dn.l),d(An)     n d(An, Dn.l) n d(An) b. move,
move.w d(An,Dn.l),d(An)     n d(An, Dn.l) n d(An) w. move,
move.l d(An,Dn.l),d(An)     n d(An, Dn.l) n d(An) l. move,

move.b d(An,Dn.l),d(An,Dn.l)
                          n d(An, Dn.l) n d(An, Dn.l) b. move,
move.w d(An,Dn.l),d(An,Dn.l)
                          n d(An, Dn.l) n d(An, Dn.l) w. move,
move.l d(An,Dn.l),d(An,Dn.l)
                          n d(An, Dn.l) n d(An, Dn.l) l. move,

move.b d(An,Dn.l),$d        n d(An, Dn.l) $ n b. move,
move.w d(An,Dn.l),$d        n d(An, Dn.l) $ n w. move,
move.l d(An,Dn.l),$d        n d(An, Dn.l) $ n l. move,

move.b $d,An                $ n An b. move,
move.w $d,An                $ n An w. move,
move.l $d,An                $ n An l. move,
```

```
move.b $d,(An)              $ n (An) b. move,
move.w $d,(An)              $ n (An) w. move,
move.l $d,(An)              $ n (An) l. move,

move.b $d,-(An)             $ n -(An) b. move,
move.w $d,-(An)             $ n -(An) w. move,
move.l $d,-(An)             $ n -(An) l. move,

move.b $d,(An)+             $ n (An)+ b. move,
move.w $d,(An)+             $ n (An)+ w. move,
move.l $d,(An)+             $ n (An)+ l. move,

move.b $d,d(An)             $ n n d(An) b. move,
move.w $d,d(An)             $ n n d(An) w. move,
move.l $d,d(An)             $ n n d(An) l. move,

move.b $d,d(An,Dn.l)        $ n n d(An, Dn.l) b. move,
move.w $d,d(An,Dn.l)        $ n n d(An, Dn.l) w. move,
move.l $d,d(An,Dn.l)        $ n n d(An, Dn.l) l. move,

move.b $d,$d                $ n $ n b. move,
move.w $d,$d                $ n $ n w. move,
move.l $d,$d                $ n $ n l. move,

move.b #i,An                # n An b. move,
move.w #i,An                # n An w. move,
move.l #i,An                # n An l. move,

move.b #i,(An)              # n (An) b. move,
move.w #i,(An)              # n (An) w. move,
move.l #i,(An)              # n (An) l. move,

move.b #i,-(An)             # n -(An) b. move,
move.w #i,-(An)             # n -(An) w. move,
move.l #i,-(An)             # n -(An) l. move,

move.b #i,(An)+             # n (An)+ b. move,
move.w #i,(An)+             # n (An)+ w. move,
move.l #i,(An)+             # n (An)+ l. move,

move.b #i,d(An)             # n n d(An) b. move,
move.w #i,d(An)             # n n d(An) w. move,
move.l #i,d(An)             # n n d(An) l. move,

move.b #i,d(An,Dn.l)        # n n d(An, Dn.l) b. move,
move.w #i,d(An,Dn.l)        # n n d(An, Dn.l) w. move,
move.l #i,d(An,Dn.l)        # n n d(An, Dn.l) l. move,
```

```
move.b #i,$d                # n $ n b. move,
move.w #i,$d                # n $ n w. move,
move.l #i,$d                # n $ n l. move,

movea.w Dn,An               Dn An w. movea,
movea.w An,An               An An w. movea,
movea.w (An),An             (An) An w. movea,
movea.w -(An),An            -(An) An w. movea,
movea.w (An)+,An            (An)+ An w. movea,
movea.w d(An),An            n d(An) An w. movea,
movea.w d(An,Dn.l),An       n d(An, Dn.l) An w. movea,
movea.w $d,An               $ n An w. movea,
movea.w $var,An             $ var An w. movea,
movea.l Dn,An               Dn An l. movea,
movea.l An,An               An An l. movea,
movea.l (An),An             (An) An l. movea,
movea.l -(An),An            -(An) An l. movea,
movea.l (An)+,An            (An)+ An l. movea,
movea.l d(An),An            n d(An) An l. movea,
movea.l d(An,Dn.l),An       n d(An, Dn.l) An l. movea,
movea.l $d,An               $ n An l. movea,
movea.l $var,An             $ var An l. movea,

move.w Dn,ccr               Dn ccr w. move,
move.w (An),ccr             (An) ccr w. move,
move.w -(An),ccr            -(An) ccr w. move,
move.w (An)+,ccr            (An)+ ccr w. move,
move.w d(An),ccr            n d(An) ccr w. move,
move.w d(An,Dn.l),ccr       n d(An, Dn.l) ccr w. move,
move.w $d,ccr               $ n ccr w. move,

move.w ccr,Dn               ccr Dn w. move,
move.w ccr,(An)             ccr (An) w. move,
move.w ccr,-(An)            ccr -(An) w. move,
move.w ccr,(An)+            ccr (An)+ w. move,
move.w ccr,d(An)            ccr n d(An) w. move,
move.w ccr,d(An,Dn.l)       ccr n d(An, Dn.l) w. move,
move.w ccr,$d               ccr $ n w. move,

move.w Dn,sr                Dn sr w. move,
move.w (An),sr              (An) sr w. move,
move.w -(An),sr             -(An) sr w. move,
move.w (An)+,sr             (An)+ sr w. move,
move.w d(An),sr             n d(An) sr w. move,
move.w d(An,Dn.l),sr        n d(An, Dn.l) sr w. move,
move.w $d,sr                $ n sr w. move,

move.w sr,Dn                sr Dn w. move,
```

```
move.w sr,(An)              sr (An) w. move,
move.w sr,-(An)             sr -(An) w. move,
move.w sr,(An)+             sr (An)+ w. move,
move.w sr,d(An)             sr n d(An) w. move,
move.w sr,d(An,Dn.l)        sr n d(An, Dn.l) w. move,
move.w sr,$d                sr $ n w. move,

move.l usp,An               usp An l. move,
move.l An,usp               An usp l. move,

movec.l sfc,Dn              sfc Dn l. movec,
movec.l Dn,sfc              Dn sfc l. movec,
movec.l dfc,Dn              dfc Dn l. movec,
movec.l Dn,dfc              Dn dfc l. movec,
movec.l vbr,Dn              vbr Dn l. movec,
movec.l Dn,vbr              Dn vbr l. movec,
movec.l Dn,usp              Dn usp l. movec,
movec.l usp,Dn              usp Dn l. movec,

movem.w Dn/Dn/An-An,-(An)   { Dn Dn An-n } -(An) w. movem,
movem.l Dn/Dn/An-An,-(An)   { Dn Dn An-n } -(An) l. movem,

movem.w Dn/Dn/An-An,(An)    { Dn Dn An-n } (An) w. movem,
movem.w Dn/Dn/An-An,d(An)   { Dn Dn An-n } n d(An) w. movem,
movem.w Dn/Dn/An-An,d(An,Dn.l)
                      { Dn Dn An-n } n d(An, Dn.l) w. movem,
movem.w Dn/Dn/An-An,$d      { Dn Dn An-n } $ n w. movem,
movem.l Dn/Dn/An-An,(An)    { Dn Dn An-n } (An) l. movem,
movem.l Dn/Dn/An-An,d(An)   { Dn Dn An-n } n d(An) l. movem,
movem.l Dn/Dn/An-An,d(An,Dn.l)
                      { Dn Dn An-n } n d(An, Dn.l) l. movem,
movem.l Dn/Dn/An-An,$d      { Dn Dn An-n } $ n l. movem,

movem.w (An)+,Dn/Dn/An-An   (An)+ { Dn Dn An-n } w. movem,
movem.l (An)+,Dn/Dn/An-An   (An)+ { Dn Dn An-n } l. movem,

movem.w (An),Dn/Dn/An-An    (An) { Dn Dn An-n } w. movem,
movem.w d(An),Dn/Dn/An-An   n d(An) { Dn Dn An-n } w. movem,
movem.w d(An,Dn.l),Dn/Dn/An-An
                      n d(An, Dn.l) { Dn Dn An-n } w. movem,
movem.w $d,Dn/Dn/An-An      $ n { Dn Dn An-n } w. movem,
movem.l (An),Dn/Dn/An-An    (An) { Dn Dn An-n } l. movem,
movem.l d(An),Dn/Dn/An-An   n d(An) { Dn Dn An-n } l. movem,
movem.l d(An,Dn.l),Dn/Dn/An-An
                      n d(An, Dn.l) { Dn Dn An-n } l. movem,
movem.l $d,Dn/Dn/An-An      $ n { Dn Dn An-n } l. movem,

movep.w Dn,d(An)            Dn n d(An) w. movep,
```

```
movep.l Dn,d(An)            Dn n d(An) l. movep,

movep.w d(An),Dn            n d(An) Dn w. movep,
movep.l d(An),Dn            n d(An) Dn l. movep,

moveq.l #i,Dn               # n Dn l. moveq,

moves.w (An),Dn             (An) Dn w. moves,
moves.w -(An),Dn            -(An) Dn w. moves,
moves.w (An)+,Dn            (An)+ Dn w. moves,
moves.w d(An),Dn            n d(An) Dn w. moves,
moves.w d(An,Dn.l),Dn       n d(An, Dn.l) Dn w. moves,
moves.w $d,Dn               $ n Dn w. moves,

moves.w Dn,(An)             Dn (An) w. moves,
moves.w Dn,-(An)            Dn -(An) w. moves,
moves.w Dn,(An)+            Dn (An)+ w. moves,
moves.w Dn,d(An)            Dn n d(An) w. moves,
moves.w Dn,d(An,Dn.l)       Dn n d(An, Dn.l) w. moves,
moves.w Dn,$d               Dn $ n w. moves,

muls.w Dn,Dn                Dn Dn w. muls,
muls.w (An),Dn              (An) Dn w. muls,
muls.w -(An),Dn             -(An) Dn w. muls,
muls.w (An)+,Dn             (An)+ Dn w. muls,
muls.w d(An),Dn             n d(An) Dn w. muls,
muls.w d(An,Dn.l),Dn        n d(An, Dn.l) Dn w. muls,
muls.w $d,Dn                $ n Dn w. muls,
muls.l Dn,Dn                Dn Dn l. muls,
muls.l (An),Dn              (An) Dn l. muls,
muls.l -(An),Dn             -(An) Dn l. muls,
muls.l (An)+,Dn             (An)+ Dn l. muls,
muls.l d(An),Dn             n d(An) Dn l. muls,
muls.l d(An,Dn.l),Dn        n d(An, Dn.l) Dn l. muls,
muls.l $d,Dn                $ n Dn l. muls,

muls.l Dn,Dn:Dn             Dn Dn: Dn l. muls,
muls.l (An),Dn:Dn           (An) Dn: Dn l. muls,
muls.l -(An),Dn:Dn          -(An) Dn: Dn l. muls,
muls.l (An)+,Dn:Dn          (An)+ Dn: Dn l. muls,
muls.l d(An),Dn:Dn          n d(An) Dn: Dn l. muls,
muls.l d(An,Dn.l),Dn:Dn     n d(An, Dn.l) Dn: Dn l. muls,
muls.l $d,Dn:Dn             $ n Dn: Dn l. muls,

mulu.w Dn,Dn                Dn Dn w. mulu,
mulu.w (An),Dn              (An) Dn w. mulu,
mulu.w -(An),Dn             -(An) Dn w. mulu,
mulu.w (An)+,Dn             (An)+ Dn w. mulu,
```

```
mulu.w d(An),Dn             n d(An) Dn w. mulu,
mulu.w d(An,Dn.l),Dn        n d(An, Dn.l) Dn w. mulu,
mulu.w $d,Dn                $ n Dn w. mulu,
mulu.l Dn,Dn                Dn Dn l. mulu,
mulu.l (An),Dn              (An) Dn l. mulu,
mulu.l -(An),Dn             -(An) Dn l. mulu,
mulu.l (An)+,Dn             (An)+ Dn l. mulu,
mulu.l d(An),Dn             n d(An) Dn l. mulu,
mulu.l d(An,Dn.l),Dn        n d(An, Dn.l) Dn l. mulu,
mulu.l $d,Dn                $ n Dn l. mulu,

mulu.l Dn,Dn:Dn             Dn Dn: Dn l. mulu,
mulu.l (An),Dn:Dn           (An) Dn: Dn l. mulu,
mulu.l -(An),Dn:Dn          -(An) Dn: Dn l. mulu,
mulu.l (An)+,Dn:Dn          (An)+ Dn: Dn l. mulu,
mulu.l d(An),Dn:Dn          n d(An) Dn: Dn l. mulu,
mulu.l d(An,Dn.l),Dn:Dn     n d(An, Dn.l) Dn: Dn l. mulu,
mulu.l $d,Dn:Dn             $ n Dn: Dn l. mulu,

mulu.w (An),Dn              (An) Dn w. mulu,
mulu.w -(An),Dn             -(An) Dn w. mulu,
mulu.w (An)+,Dn             (An)+ Dn w. mulu,
mulu.w d(An),Dn             n d(An) Dn w. mulu,
mulu.w d(An,Dn.l),Dn        n d(An, Dn.l) Dn w. mulu,
mulu.w $d,Dn                $ n Dn w. mulu,

nbcd.b,Dn                   Dn b. nbcd,
nbcd.b,(An)                 (An) b. nbcd,
nbcd.b,-(An)                -(An) b. nbcd,
nbcd.b,(An)+                (An)+ b. nbcd,
nbcd.b,d(An)                n d(An) b. nbcd,
nbcd.b,d(An,Dn.l)           n d(An, Dn.l) b. nbcd,
nbcd.b $d                   $ n b. nbcd,

neg.b Dn                    Dn b. neg,
neg.b (An)                  (An) b. neg,
neg.b -(An)                 -(An) b. neg,
neg.b (An)+                 (An)+ b. neg,
neg.b d(An)                 n d(An) b. neg,
neg.b d(An,Dn.l)            n d(An, Dn.l) b. neg,
neg.b $d                    $ n b. neg,
neg.w Dn                    Dn w. neg,
neg.w (An)                  (An) w. neg,
neg.w -(An)                 -(An) w. neg,
neg.w (An)+                 (An)+ w. neg,
neg.w d(An)                 n d(An) w. neg,
neg.w d(An,Dn.l)            n d(An, Dn.l) w. neg,
neg.w $d                    $ n w. neg,
```

```
neg.l Dn                    Dn l. neg,
neg.l (An)                  (An) l. neg,
neg.l -(An)                 -(An) l. neg,
neg.l (An)+                 (An)+ l. neg,
neg.l d(An)                 n d(An) l. neg,
neg.l d(An,Dn.l)            n d(An, Dn.l) l. neg,
neg.l $d                    $ n l. neg,

negx.b Dn                   Dn b. negx,
negx.b (An)                 (An) b. negx,
negx.b -(An)                -(An) b. negx,
negx.b (An)+                (An)+ b. negx,
negx.b d(An)                n d(An) b. negx,
negx.b d(An,Dn.l)           n d(An, Dn.l) b. negx,
negx.b $d                   $ n b. negx,
negx.w Dn                   Dn w. negx,
negx.w (An)                 (An) w. negx,
negx.w -(An)                -(An) w. negx,
negx.w (An)+                (An)+ w. negx,
negx.w d(An)                n d(An) w. negx,
negx.w d(An,Dn.l)           n d(An, Dn.l) w. negx,
negx.w $d                   $ n w. negx,
negx.l Dn                   Dn l. negx,
negx.l (An)                 (An) l. negx,
negx.l -(An)                -(An) l. negx,
negx.l (An)+                (An)+ l. negx,
negx.l d(An)                n d(An) l. negx,
negx.l d(An,Dn.l)           n d(An, Dn.l) l. negx,
negx.l $d                   $ n l. negx,

nop                         nop,

not.b Dn                    Dn b. not,
not.b (An)                  (An) b. not,
not.b -(An)                 -(An) b. not,
not.b (An)+                 (An)+ b. not,
not.b d(An)                 n d(An) b. not,
not.b d(An,Dn.l)            n d(An, Dn.l) b. not,
not.b $d                    $ n b. not,
not.w Dn                    Dn w. not,
not.w (An)                  (An) w. not,
not.w -(An)                 -(An) w. not,
not.w (An)+                 (An)+ w. not,
not.w d(An)                 n d(An) w. not,
not.w d(An,Dn.l)            n d(An, Dn.l) w. not,
not.w $d                    $ n w. not,
not.l Dn                    Dn l. not,
not.l (An)                  (An) l. not,
```

```
not.l -(An)                 -(An) l. not,
not.l (An)+                 (An)+ l. not,
not.l d(An)                 n d(An) l. not,
not.l d(An,Dn.l)            n d(An, Dn.l) l. not,
not.l $d                    $ n l. not,

or.b Dn,Dn                  Dn Dn b. or,
or.b (An),Dn                (An) Dn b. or,
or.b -(An),Dn               -(An) Dn b. or,
or.b (An)+,Dn               (An)+ Dn b. or,
or.b d(An),Dn               n d(An) Dn b. or,
or.b d(An,Dn.l),Dn          n d(An, Dn.l) Dn b. or,
or.b $d,Dn                  $ n Dn b. or,
or.w Dn,Dn                  Dn Dn w. or,
or.w (An),Dn                (An) Dn w. or,
or.w -(An),Dn               -(An) Dn w. or,
or.w (An)+,Dn               (An)+ Dn w. or,
or.w d(An),Dn               n d(An) Dn w. or,
or.w d(An,Dn.l),Dn          n d(An, Dn.l) Dn w. or,
or.w $d,Dn                  $ n Dn w. or,
or.l Dn,Dn                  Dn Dn l. or,
or.l (An),Dn                (An) Dn l. or,
or.l -(An),Dn               -(An) Dn l. or,
or.l (An)+,Dn               (An)+ Dn l. or,
or.l d(An),Dn               n d(An) Dn l. or,
or.l d(An,Dn.l),Dn          n d(An, Dn.l) Dn l. or,
or.l $d,Dn                  $ n Dn l. or,

or.b Dn,Dn                  Dn Dn b. or,
or.b Dn,(An)                Dn (An) b. or,
or.b Dn,-(An)               Dn -(An) b. or,
or.b Dn,(An)+               Dn (An)+ b. or,
or.b Dn,d(An)               Dn n d(An) b. or,
or.b Dn,d(An,Dn.l)          Dn n d(An, Dn.l) b. or,
or.b Dn,$d                  Dn $ n b. or,
or.w Dn,Dn                  Dn Dn w. or,
or.w Dn,(An)                Dn (An) w. or,
or.w Dn,-(An)               Dn -(An) w. or,
or.w Dn,(An)+               Dn (An)+ w. or,
or.w Dn,d(An)               Dn n d(An) w. or,
or.w Dn,d(An,Dn.l)          Dn n d(An, Dn.l) w. or,
or.w Dn,$d                  Dn $ n w. or,
or.l Dn,Dn                  Dn Dn l. or,
or.l Dn,(An)                Dn (An) l. or,
or.l Dn,-(An)               Dn -(An) l. or,
or.l Dn,(An)+               Dn (An)+ l. or,
or.l Dn,d(An)               Dn n d(An) l. or,
or.l Dn,d(An,Dn.l)          Dn n d(An, Dn.l) l. or,
```

```
or.l Dn,$d                  Dn $ n l. or,

ori.b #i,Dn                 # n Dn b. ori,
ori.b #i,(An)               # n (An) b. ori,
ori.b #i,-(An)              # n -(An) b. ori,
ori.b #i,(An)+              # n (An)+ b. ori,
ori.b #i,d(An)              # n n d(An) b. ori,
ori.b #i,d(An,Dn.l)         # n n d(An, Dn.l) b. ori,
ori.b #i,$d                 # n $ n b. ori,
ori.w #i,Dn                 # n Dn w. ori,
ori.w #i,(An)               # n (An) w. ori,
ori.w #i,-(An)              # n -(An) w. ori,
ori.w #i,(An)+              # n (An)+ w. ori,
ori.w #i,d(An)              # n n d(An) w. ori,
ori.w #i,d(An,Dn.l)         # n n d(An, Dn.l) w. ori,
ori.w #i,$d                 # n $ n w. ori,
ori.l #i,Dn                 # n Dn l. ori,
ori.l #i,(An)               # n (An) l. ori,
ori.l #i,-(An)              # n -(An) l. ori,
ori.l #i,(An)+              # n (An)+ l. ori,
ori.l #i,d(An)              # n n d(An) l. ori,
ori.l #i,d(An,Dn.l)         # n n d(An, Dn.l) l. ori,
ori.l #i,$d                 # n $ n l. ori,

ori.b #i,ccr                # n ccr b. ori,

ori.w #i,sr                 # n sr w. ori,

pea.l (An)                  (An) l. pea,
pea.l d(An)                 n d(An) l. pea,
pea.l d(An,Dn.l)            n d(An, Dn.l) l. pea,
pea.l $d                    $ n l. pea,

reset                       reset,

rol.b Dn,Dn                 Dn Dn b. rol,
rol.b #i,Dn                 # n Dn b. rol,
rol.w Dn,Dn                 Dn Dn w. rol,
rol.w #i,Dn                 # n Dn w. rol,
rol.l Dn,Dn                 Dn Dn l. rol,
rol.l #i,Dn                 # n Dn l. rol,

rol.b Dn                    # 1 Dn b. rol,
rol.b (An)                  # 1 (An) b. rol,
rol.b -(An)                 # 1 -(An) b. rol,
rol.b (An)+                 # 1 (An)+ b. rol,
rol.b d(An)                 # 1 n d(An) b. rol,
rol.b d(An,Dn.l)            # 1 n d(An, Dn.l) b. rol,
```

```
rol.b $d                    # 1 $ n b. rol,
rol.b $var                  # 1 $ var b. rol,
rol.w Dn                    # 1 Dn w. rol,
rol.w (An)                  # 1 (An) w. rol,
rol.w -(An)                 # 1 -(An) w. rol,
rol.w (An)+                 # 1 (An)+ w. rol,
rol.w d(An)                 # 1 n d(An) w. rol,
rol.w d(An,Dn.l)            # 1 n d(An, Dn.l) w. rol,
rol.w $d                    # 1 $ n w. rol,
rol.w $var                  # 1 $ var w. rol,
rol.l Dn                    # 1 Dn l. rol,
rol.l (An)                  # 1 (An) l. rol,
rol.l -(An)                 # 1 -(An) l. rol,
rol.l (An)+                 # 1 (An)+ l. rol,
rol.l d(An)                 # 1 n d(An) l. rol,
rol.l d(An,Dn.l)            # 1 n d(An, Dn.l) l. rol,
rol.l $d                    # 1 $ n l. rol,
rol.l $var                  # 1 $ var l. rol,

ror.b Dn,Dn                 Dn Dn b. ror,
ror.b #i,Dn                 # n Dn b. ror,
ror.w Dn,Dn                 Dn Dn w. ror,
ror.w #i,Dn                 # n Dn w. ror,
ror.l Dn,Dn                 Dn Dn l. ror,
ror.l #i,Dn                 # n Dn l. ror,

ror.b Dn                    # 1 Dn b. ror,
ror.b (An)                  # 1 (An) b. ror,
ror.b -(An)                 # 1 -(An) b. ror,
ror.b (An)+                 # 1 (An)+ b. ror,
ror.b d(An)                 # 1 n d(An) b. ror,
ror.b d(An,Dn.l)            # 1 n d(An, Dn.l) b. ror,
ror.b $d                    # 1 $ n b. ror,
ror.b $var                  # 1 $ var b. ror,
ror.w Dn                    # 1 Dn w. ror,
ror.w (An)                  # 1 (An) w. ror,
ror.w -(An)                 # 1 -(An) w. ror,
ror.w (An)+                 # 1 (An)+ w. ror,
ror.w d(An)                 # 1 n d(An) w. ror,
ror.w d(An,Dn.l)            # 1 n d(An, Dn.l) w. ror,
ror.w $d                    # 1 $ n w. ror,
ror.w $var                  # 1 $ var w. ror,
ror.l Dn                    # 1 Dn l. ror,
ror.l (An)                  # 1 (An) l. ror,
ror.l -(An)                 # 1 -(An) l. ror,
ror.l (An)+                 # 1 (An)+ l. ror,
ror.l d(An)                 # 1 n d(An) l. ror,
ror.l d(An,Dn.l)            # 1 n d(An, Dn.l) l. ror,
```

```
ror.l $d                    # 1 $ n l. ror,
ror.l $var                  # 1 $ var l. ror,

roxl.b Dn,Dn                Dn Dn b. roxl,
roxl.b #i,Dn                # n Dn b. roxl,
roxl.w Dn,Dn                Dn Dn w. roxl,
roxl.w #i,Dn                # n Dn w. roxl,
roxl.l Dn,Dn                Dn Dn l. roxl,
roxl.l #i,Dn                # n Dn l. roxl,

roxl.b Dn                   # 1 Dn b. roxl,
roxl.b (An)                 # 1 (An) b. roxl,
roxl.b -(An)                # 1 -(An) b. roxl,
roxl.b (An)+                # 1 (An)+ b. roxl,
roxl.b d(An)                # 1 n d(An) b. roxl,
roxl.b d(An,Dn.l)           # 1 n d(An, Dn.l) b. roxl,
roxl.b $d                   # 1 $ n b. roxl,
roxl.b $var                 # 1 $ var b. roxl,
roxl.w Dn                   # 1 Dn w. roxl,
roxl.w (An)                 # 1 (An) w. roxl,
roxl.w -(An)                # 1 -(An) w. roxl,
roxl.w (An)+                # 1 (An)+ w. roxl,
roxl.w d(An)                # 1 n d(An) w. roxl,
roxl.w d(An,Dn.l)           # 1 n d(An, Dn.l) w. roxl,
roxl.w $d                   # 1 $ n w. roxl,
roxl.w $var                 # 1 $ var w. roxl,
roxl.l Dn                   # 1 Dn l. roxl,
roxl.l (An)                 # 1 (An) l. roxl,
roxl.l -(An)                # 1 -(An) l. roxl,
roxl.l (An)+                # 1 (An)+ l. roxl,
roxl.l d(An)                # 1 n d(An) l. roxl,
roxl.l d(An,Dn.l)           # 1 n d(An, Dn.l) l. roxl,
roxl.l $d                   # 1 $ n l. roxl,
roxl.l $var                 # 1 $ var l. roxl,

roxr.b Dn,Dn                Dn Dn b. roxr,
roxr.b #i,Dn                # n Dn b. roxr,
roxr.w Dn,Dn                Dn Dn w. roxr,
roxr.w #i,Dn                # n Dn w. roxr,
roxr.l Dn,Dn                Dn Dn l. roxr,
roxr.l #i,Dn                # n Dn l. roxr,
roxr.b Dn                   # 1 Dn b. roxr,
roxr.b (An)                 # 1 (An) b. roxr,
roxr.b -(An)                # 1 -(An) b. roxr,
roxr.b (An)+                # 1 (An)+ b. roxr,
roxr.b d(An)                # 1 n d(An) b. roxr,
roxr.b d(An,Dn.l)           # 1 n d(An, Dn.l) b. roxr,
roxr.b $d                   # 1 $ n b. roxr,
```

```
roxr.b $var                 # 1 $ var b. roxr,
roxr.w Dn                   # 1 Dn w. roxr,
roxr.w (An)                 # 1 (An) w. roxr,
roxr.w -(An)                # 1 -(An) w. roxr,
roxr.w (An)+                # 1 (An)+ w. roxr,
roxr.w d(An)                # 1 n d(An) w. roxr,
roxr.w d(An,Dn.l)           # 1 n d(An, Dn.l) w. roxr,
roxr.w $d                   # 1 $ n w. roxr,
roxr.w $var                 # 1 $ var w. roxr,
roxr.l Dn                   # 1 Dn l. roxr,
roxr.l (An)                 # 1 (An) l. roxr,
roxr.l -(An)                # 1 -(An) l. roxr,
roxr.l (An)+                # 1 (An)+ l. roxr,
roxr.l d(An)                # 1 n d(An) l. roxr,
roxr.l d(An,Dn.l)           # 1 n d(An, Dn.l) l. roxr,
roxr.l $d                   # 1 $ n l. roxr,
roxr.l $var                 # 1 $ var l. roxr,

rte                         rte,

rtr                         rtr,

rts                         rts,

sbcd.b Dn,Dn                Dn Dn b. sbcd,

sbcd.b -(An),-(An)          -(An) -(An) b. sbcd,

shi                         shi,
sls                         sls,
scc                         scc,
scs                         scs,
sne                         sne,
seq                         seq,
svc                         svc,
svs                         svs,
spl                         spl,
smi                         smi,
sge                         sge,
slt                         slt,
sgt                         sgt,
sle                         sle,

stop #i                     # n stop,

sub.b Dn,Dn                 Dn Dn b. sub,
sub.b (An),Dn               (An) Dn b. sub,
sub.b -(An),Dn              -(An) Dn b. sub,
```

```
sub.b (An)+,Dn              (An)+ Dn b. sub,
sub.b d(An),Dn              n d(An) Dn b. sub,
sub.b d(An,Dn.l),Dn         n d(An, Dn.l) Dn b. sub,
sub.b $d,Dn                 $ n Dn b. sub,
sub.b $var,Dn               $ var Dn b. sub,
sub.b $d,Dn                 $ n Dn b. sub,
sub.w Dn,Dn                 Dn Dn w. sub,
sub.w (An),Dn               (An) Dn w. sub,
sub.w -(An),Dn              -(An) Dn w. sub,
sub.w (An)+,Dn              (An)+ Dn w. sub,
sub.w d(An),Dn              n d(An) Dn w. sub,
sub.w d(An,Dn.l),Dn         n d(An, Dn.l) Dn w. sub,
sub.w $d,Dn                 $ n Dn w. sub,
sub.w $var,Dn               $ var Dn w. sub,
sub.w $d,Dn                 $ n Dn w. sub,
sub.l Dn,Dn                 Dn Dn l. sub,
sub.l (An),Dn               (An) Dn l. sub,
sub.l -(An),Dn              -(An) Dn l. sub,
sub.l (An)+,Dn              (An)+ Dn l. sub,
sub.l d(An),Dn              n d(An) Dn l. sub,
sub.l d(An,Dn.l),Dn         n d(An, Dn.l) Dn l. sub,
sub.l $d,Dn                 $ n Dn l. sub,
sub.l $var,Dn               $ var Dn l. sub,
sub.l $d,Dn                 $ n Dn l. sub,
sub.b Dn,Dn                 Dn Dn b. sub,
sub.b Dn,(An)               Dn (An) b. sub,
sub.b Dn,-(An)              Dn -(An) b. sub,
sub.b Dn,(An)+              Dn (An)+ b. sub,
sub.b Dn,d(An)              Dn n d(An) b. sub,
sub.b Dn,d(An,Dn.l)         Dn n d(An, Dn.l) b. sub,
sub.b Dn,$d                 Dn $ n b. sub,
sub.b Dn,$var               Dn $ var b. sub,
sub.b Dn,$d                 Dn $ n b. sub,
sub.w Dn,Dn                 Dn Dn w. sub,
sub.w Dn,(An)               Dn (An) w. sub,
sub.w Dn,-(An)              Dn -(An) w. sub,
sub.w Dn,(An)+              Dn (An)+ w. sub,
sub.w Dn,d(An)              Dn n d(An) w. sub,
sub.w Dn,d(An,Dn.l)         Dn n d(An, Dn.l) w. sub,
sub.w Dn,$d                 Dn $ n w. sub,
sub.w Dn,$var               Dn $ var w. sub,
sub.w Dn,$d                 Dn $ n w. sub,
sub.l Dn,Dn                 Dn Dn l. sub,
sub.l Dn,(An)               Dn (An) l. sub,
sub.l Dn,-(An)              Dn -(An) l. sub,
sub.l Dn,(An)+              Dn (An)+ l. sub,
sub.l Dn,d(An)              Dn n d(An) l. sub,
sub.l Dn,d(An,Dn.l)         Dn n d(An, Dn.l) l. sub,
```

```
sub.l Dn,$d                 Dn $ n l. sub,
sub.l Dn,$var               Dn $ var l. sub,
sub.l Dn,$d                 Dn $ n l. sub,

suba.w An,An                An An w. suba,
suba.w (An),An              (An) An w. suba,
suba.w -(An),An             -(An) An w. suba,
suba.w (An)+,An             (An)+ An w. suba,
suba.w d(An),An             n d(An) An w. suba,
suba.w d(An,Dn.l),An        n d(An, Dn.l) An w. suba,
suba.w $d,An                $ n An w. suba,
suba.w $var,An              $ var An w. suba,
suba.w $d,An                $ n An w. suba,
suba.l An,An                An An l. suba,
suba.l (An),An              (An) An l. suba,
suba.l -(An),An             -(An) An l. suba,
suba.l (An)+,An             (An)+ An l. suba,
suba.l d(An),An             n d(An) An l. suba,
suba.l d(An,Dn.l),An        n d(An, Dn.l) An l. suba,
suba.l $d,An                $ n An l. suba,
suba.l $var,An              $ var An l. suba,
suba.l $d,An                $ n An l. suba,

subi.b #i,Dn                # n Dn b. subi,
subi.b #i,(An)              # n (An) b. subi,
subi.b #i,-(An)             # n -(An) b. subi,
subi.b #i,(An)+             # n (An)+ b. subi,
subi.b #i,d(An)             # n n d(An) b. subi,
subi.b #i,d(An,Dn.l)        # n n d(An, Dn.l) b. subi,
subi.b #i,$d                # n $ n b. subi,
subi.b #i,$var              # n $ var b. subi,
subi.b #i,$d                # n $ n b. subi,
subi.w #i,Dn                # n Dn w. subi,
subi.w #i,(An)              # n (An) w. subi,
subi.w #i,-(An)             # n -(An) w. subi,
subi.w #i,(An)+             # n (An)+ w. subi,
subi.w #i,d(An)             # n n d(An) w. subi,
subi.w #i,d(An,Dn.l)        # n n d(An, Dn.l) w. subi,
subi.w #i,$d                # n $ n w. subi,
subi.w #i,$var              # n $ var w. subi,
subi.w #i,$d                # n $ n w. subi,
subi.l #i,Dn                # n Dn l. subi,
subi.l #i,(An)              # n (An) l. subi,
subi.l #i,-(An)             # n -(An) l. subi,
subi.l #i,(An)+             # n (An)+ l. subi,
subi.l #i,d(An)             # n n d(An) l. subi,
subi.l #i,d(An,Dn.l)        # n n d(An, Dn.l) l. subi,
subi.l #i,$d                # n $ n l. subi,
```

```
subi.l #i,$var              # n $ var l. subi,
subi.l #i,$d                # n $ n l. subi,

subq.b #i,Dn                # n Dn b. subq,
subq.b #i,(An)              # n (An) b. subq,
subq.b #i,-(An)             # n -(An) b. subq,
subq.b #i,(An)+             # n (An)+ b. subq,
subq.b #i,d(An)             # n n d(An) b. subq,
subq.b #i,d(An,Dn.l)        # n n d(An, Dn.l) b. subq,
subq.b #i,$d                # n $ n b. subq,
subq.b #i,$var              # n $ var b. subq,
subq.b #i,$d                # n $ n b. subq,
subq.w #i,Dn                # n Dn w. subq,
subq.w #i,(An)              # n (An) w. subq,
subq.w #i,-(An)             # n -(An) w. subq,
subq.w #i,(An)+             # n (An)+ w. subq,
subq.w #i,d(An)             # n n d(An) w. subq,
subq.w #i,d(An,Dn.l)        # n n d(An, Dn.l) w. subq,
subq.w #i,$d                # n $ n w. subq,
subq.w #i,$var              # n $ var w. subq,
subq.w #i,$d                # n $ n w. subq,
subq.l #i,Dn                # n Dn l. subq,
subq.l #i,(An)              # n (An) l. subq,
subq.l #i,-(An)             # n -(An) l. subq,
subq.l #i,(An)+             # n (An)+ l. subq,
subq.l #i,d(An)             # n n d(An) l. subq,
subq.l #i,d(An,Dn.l)        # n n d(An, Dn.l) l. subq,
subq.l #i$d                 # n $ n l. subq,
subq.l #i,$var              # n $ var l. subq,
subq.l #i,$d                # n $ n l. subq,

subx.b Dn,Dn                Dn Dn b. subx,
subx.w Dn,Dn                Dn Dn w. subx,
subx.l Dn,Dn                Dn Dn l. subx,
subx.b -(An),-(An)          -(An) -(An) b. subx,
subx.w -(An),-(An)          -(An) -(An) w. subx,
subx.l -(An),-(An)          -(An) -(An) l. subx,

swap.w Dn                   Dn w. swap,

tas.b Dn                    Dn b. tas,
tas.b (An)                  (An) b. tas,
tas.b -(An)                 -(An) b. tas,
tas.b (An)+                 (An)+ b. tas,
tas.b d(An)                 n d(An) b. tas,
tas.b d(An,Dn.l)            n d(An, Dn.l) b. tas,
tas.b $d                    $ n b. tas,
tas.b $var                  $ var b. tas,
```

```
tas.b $d                    $ n b. tas,

tbls.b (An),Dn              (An) Dn b. tbls
tbls.b d(An),Dn             n d(An) Dn b. tbls
tbls.b d(An,Dn.l),Dn        n d(An, Dn.l) Dn b. tbls
tbls.b $d,Dn                $ n Dn b. tbls
tbls.b $var,Dn              $ var Dn b. tbls
tbls.b $d,Dn                $ n Dn b. tbls
tbls.w (An),Dn              (An) Dn w. tbls
tbls.w d(An),Dn             n d(An) Dn w. tbls
tbls.w d(An,Dn.l),Dn        n d(An, Dn.l) Dn w. tbls
tbls.w $d,Dn                $ n Dn w. tbls
tbls.w $var,Dn              $ var Dn w. tbls
tbls.w $d,Dn                $ n Dn w. tbls
tbls.l (An),Dn              (An) Dn l. tbls
tbls.l d(An),Dn             n d(An) Dn l. tbls
tbls.l d(An,Dn.l),Dn        n d(An, Dn.l) Dn l. tbls
tbls.l $d,Dn                $ n Dn l. tbls
tbls.l $var,Dn              $ var Dn l. tbls
tbls.l $d,Dn                $ n Dn l. tbls

tblsn.b (An),Dn             (An) Dn b. tblsn
tblsn.b d(An),Dn            n d(An) Dn b. tblsn
tblsn.b d(An,Dn.l),Dn       n d(An, Dn.l) Dn b. tblsn
tblsn.b $d,Dn               $ n Dn b. tblsn
tblsn.b $var,Dn             $ var Dn b. tblsn
tblsn.b $d,Dn               $ n Dn b. tblsn
tblsn.w (An),Dn             (An) Dn w. tblsn
tblsn.w d(An),Dn            n d(An) Dn w. tblsn
tblsn.w d(An,Dn.l),Dn       n d(An, Dn.l) Dn w. tblsn
tblsn.w $d,Dn               $ n Dn w. tblsn
tblsn.w $var,Dn             $ var Dn w. tblsn
tblsn.w $d,Dn               $ n Dn w. tblsn
tblsn.l (An),Dn             (An) Dn l. tblsn
tblsn.l d(An),Dn            n d(An) Dn l. tblsn
tblsn.l d(An,Dn.l),Dn       n d(An, Dn.l) Dn l. tblsn
tblsn.l $d,Dn               $ n Dn l. tblsn
tblsn.l $var,Dn             $ var Dn l. tblsn
tblsn.l $d,Dn               $ n Dn l. tblsn

tbls.b Dn:Dn,Dn             Dn: Dn Dn b. tbls

tblsn.b Dn:Dn,Dn            Dn: Dn Dn b. tblsn

tblu.b (An),Dn              (An) Dn b. tblu
tblu.b d(An),Dn             n d(An) Dn b. tblu
tblu.b d(An,Dn.l),Dn        n d(An, Dn.l) Dn b. tblu
tblu.b $d,Dn                $ n Dn b. tblu
```

```
tblu.b $var,Dn              $ var Dn b. tblu
tblu.b $d,Dn                $ n Dn b. tblu
tblu.w (An),Dn              (An) Dn w. tblu
tblu.w d(An),Dn             n d(An) Dn w. tblu
tblu.w d(An,Dn.l),Dn        n d(An, Dn.l) Dn w. tblu
tblu.w $d,Dn                $ n Dn w. tblu
tblu.w $var,Dn              $ var Dn w. tblu
tblu.w $d,Dn                $ n Dn w. tblu
tblu.l (An),Dn              (An) Dn l. tblu
tblu.l d(An),Dn             n d(An) Dn l. tblu
tblu.l d(An,Dn.l),Dn        n d(An, Dn.l) Dn l. tblu
tblu.l $d,Dn                $ n Dn l. tblu
tblu.l $var,Dn              $ var Dn l. tblu
tblu.l $d,Dn                $ n Dn l. tblu

tblun.b (An),Dn             (An) Dn b. tblun
tblun.b d(An),Dn            n d(An) Dn b. tblun
tblun.b d(An,Dn.l),Dn       n d(An, Dn.l) Dn b. tblun
tblun.b $d,Dn               $ n Dn b. tblun
tblun.b $var,Dn             $ var Dn b. tblun
tblun.b $d,Dn               $ n Dn b. tblun
tblun.w (An),Dn             (An) Dn w. tblun
tblun.w d(An),Dn            n d(An) Dn w. tblun
tblun.w d(An,Dn.l),Dn       n d(An, Dn.l) Dn w. tblun
tblun.w $d,Dn               $ n Dn w. tblun
tblun.w $var,Dn             $ var Dn w. tblun
tblun.w $d,Dn               $ n Dn w. tblun
tblun.l (An),Dn             (An) Dn l. tblun
tblun.l d(An),Dn            n d(An) Dn l. tblun
tblun.l d(An,Dn.l),Dn       n d(An, Dn.l) Dn l. tblun
tblun.l $d,Dn               $ n Dn l. tblun
tblun.l $var,Dn             $ var Dn l. tblun
tblun.l $d,Dn               $ n Dn l. tblun

tblu.b Dn:Dn,Dn             Dn: Dn Dn b. tblu

tblun.b Dn:Dn,Dn            Dn: Dn Dn b. tblun

trap #i                     # n trap,

traphi                      traphi,
trapls                      trapls,
trapcc                      trapcc,
trapcs                      trapcs,
trapne                      trapne,
trapeq                      trapeq,
trapvc                      trapvc,
trapvs                      trapvs,
```

```
trappl                  trappl,
trapmi                  trapmi,
trapge                  trapge,
traplt                  traplt,
trapgt                  trapgt,
traple                  traple,
trapv                   trapv,

tst.b Dn                Dn b. tst,
tst.b (An)              (An) b. tst,
tst.b -(An)             -(An) b. tst,
tst.b (An)+             (An)+ b. tst,
tst.b d(An)             n d(An) b. tst,
tst.b d(An,Dn.l)        n d(An, Dn.l) b. tst,
tst.b $d                $ n b. tst,
tst.b $var              $ var b. tst,
tst.b $d                $ n b. tst,
tst.w Dn                Dn w. tst,
tst.w (An)              (An) w. tst,
tst.w -(An)             -(An) w. tst,
tst.w (An)+             (An)+ w. tst,
tst.w d(An)             n d(An) w. tst,
tst.w d(An,Dn.l)        n d(An, Dn.l) w. tst,
tst.w $d                $ n w. tst,
tst.w $var              $ var w. tst,
tst.w $d                $ n w. tst,
tst.l Dn                Dn l. tst,
tst.l (An)              (An) l. tst,
tst.l -(An)             -(An) l. tst,
tst.l (An)+             (An)+ l. tst,
tst.l d(An)             n d(An) l. tst,
tst.l d(An,Dn.l)        n d(An, Dn.l) l. tst,
tst.l $d                $ n l. tst,
tst.l $var              $ var l. tst,
tst.l $d                $ n l. tst,

unlk An                 An unlk,
```

Switching between Forth and Assembler

The compiler allows you to add in-line assembler inside colon definitions, and to add high level phrases inside code definitions.

Inline assembler code can be compiled inside a colon definition using **[ASM** and **ASM]**. Use these in the form:

```
: <name>
  ... [ASM <assembler-code> ASM] ...
;
```

High level code can be compiled inside a CODE definition using **[FORTH** and **FORTH]**. Use these in the form:

```
CODE <name>
  ... [FORTH <high-level> FORTH] ...
END-CODE
```

Note that the optimiser is not flushed by the switches into assembler. This can (and should) be achieved by placing **[O/F]** before **[ASM** and **FORTH]**.

Glossary

This glossary details the words provided within the cross-assembler to control the use of the assembler.

;CODE --
"semi-code"

A defining word used in the form:

```
:<namex>CREATE...;CODE...END-CODE
```

Stops compilation, and enables the assembler. This word is used with **CREATE** to produce defining words whose run-time portion is written in code, in the same way that **CREATE ... DOES>** is used to create high level defining words .

The data structure is defined between **CREATE** and **;CODE** and the run-time action is defined between **;CODE** and **END-CODE**. The current value of the data stack pointer is saved by **;CODE** for later use by **END-CODE** for error checking. When **<namex>** executes the address of the data area will be found on the processor stack, from which it must be removed.

ASMCODE --
"asm-code"

Starts a section of assembler code and turns on the assembler, but without generating a dictionary header. This action is particularly useful for generating the start-up code. Examples of this can be found in CODE68K.FTH.

CODE --
"code"

A defining word used in the form:

```
CODE <name> ... END-CODE
```

Creates a dictionary entry for **<name>** to be defined by a following sequence of assembly language words. Words thus defined are called code definitions. **CODE** stores the current data stack pointer for later error checking by **END-CODE**.

END-CODE --
"end-code"

Terminates a code definition and checks the data stack pointer against the value stored when **;CODE or CODE** is executed. The assembler is disabled. See: **CODE ;CODE**

IS-ACTION-OF addr --
"is action of"

Used to tell the cross-compiler that the given address is to be used as the run time action of the word whose name follows. Usually found in code definitions, but can also be used for high-level definitions. For example:

```
ASMCODE
HERE IS-ACTION-OF CONSTANT
  …
END-CODE

ASSEMBLER
HERE IS-ACTION-OF <<high-level-definer>>
DODOES BRA,
END-CODE
] … EXIT [
```

PROC -- ; PROC <label-name>
"proc"

Starts a section of assembler code and turns on the assembler, defining a label. This action is particularly useful for generating interrupts or shared subroutines. Examples of this can be found in CODE68K.FTH.

ASSEMBLER --
"assembler"

Tells the compiler that the definitions that follow are extensions to the assembler.

TARGET --
"target"

Tells the compiler that the definitions that follow are target definitions. This directive is used to finish cross compiler extension started by **INTERPRETER COMPILER** or **ASSEMBLER**.

MASM: **--**
“m-colon”

Starts a macro definition.

;MASM **--**
“semi-colon-m”

Ends a macro definition

[ASM --
“bracket-asm”

Start an assembler fragment inside a colon definition.

```
: <name>
  ... [ASM <assembler-code> ASM] ...
;
```

ASM] --
“asm-bracket”

End an assembler fragment inside a colon definition.

```
: <name>
  ... [ASM <assembler-code> ASM] ...
;
```

[FORTH --
“bracket-forth”

Start a Forth fragment inside a code definition.

```
CODE <name>
  ... [FORTH <high-level> FORTH] ...
END-CODE
```

FORTH] --
“bracket-forth”

End a Forth fragment inside a code definition.

```
CODE <name>
  ... [FORTH <high-level> FORTH] ...
END-CODE
```

.

4 68000 assembler errors

Errors 32..47, 144...159

Error code	Error
033/$021	Index register out of order, or too many operands, or previous opcode.misspelled.
034/$022	Illegal source addressing mode.
035/$023	Illegal destination addressing mode.
036/$024	Illegal addressing mode.
037/$025	Offset or data out of range -128..+127 or illegal.
038/$026	Quick data out of range 1..8.
039/$027	Incorrect data size.
040/$028	Shift count out of range 1..8.
041/$029	Shift count out of range 1..64.
042/$02A	Trap number out of range 0..15.
043/$02B	Invalid data for MOVEM.
044/$02C	Not valid for this processor type.
045/$02D	INTERNAL SOFTWARE ERROR - report to MPE.
046/$02E	Unconsumed reference.
047/$02F	Code error, stack depth changed. This is a general catch all error check performed by **END-CODE**.
145/$091	Word offset not in range -32768..+32767.
146/$092	Breakpoint out of range 0..7.
147/$093	32-bit offsets not available on this processor type.

148/$094	Cannot define remainder/high register for word form.
149/$096	Remainder/high register must be specified.
150/$097	Remainder & quotient or high & low in same register.

Table 5: 68000 assembler error messages

5 68000 interrupts

This chapter describes how to write interrupt handlers in both Forth and assembler. It details how to set up and control interrupt handlers. The first part of the chapter describes how to set up and use the interrupts, and the second part provides more technical detail.

Interrupts on the 68000

When an interrupt occurs and is accepted, a 68000 processor branches through a location (a `vector') in low memory. The vector table starts at address 0, the vector used being dependent on the source of the interrupt. The code at the address stored in the vector table is then executed. For more information on interrupts for the 68000, refer to the processor's user guide.

Writing Forth interrupt handlers

A Forth interrupt service routine (ISR) is just like any other Forth word. It can therefore be tested and debugged like a normal Forth word. Only when the word is fully tested need it be assigned to an interrupt.

Setting an interrupt

An interrupt is set by using the **DEFER**red words in **Table 6**. For example, suppose you have an ADC that generates an interrupt on the level 2 interrupt line. If you wish to run your word **A/D-READY** once the A/D has finished its conversion, you need to assign an action to the deferred word **AVEC2-ISR**. Therefore your source code should read:

```
ASSIGN A/D-READY TO-DO AVEC2-ISR
```

Some common problems

There are a few common problems that might cause an interrupt not to work correctly:

- a stack fault
- the source is not cleared
- the interrupts are not enabled

Stack faults

An interrupt service routine can use the stack while it is executing, but must clear up the stack before returning from the interrupt. The normal symptom of a stack fault is that the

interrupt handler runs, but then the target board crashes either immediately or after a length of time.

Source is not cleared

Once an interrupt handler is triggered by an interrupt, the source of the interrupt must be told that the interrupt is being serviced. If this is not done, the source of the interrupt will carry on generating interrupts. Normally this appears as the interrupt handler executing once and then the target board `locking'.

Interrupts are not enabled

Interrupts need to be enabled with **EI** before any interrupts will be serviced. The vectors must be set up or the interrupt handler assigned to the deferred word before the interrupts are enabled.

Writing assembler interrupt handlers

Writing an interrupt service routine in assembler is straightforward. The code can be written in the form shown below, which is taken from the serial line driver in 68307I.FTH in the ROM\DRIVERS subdirectory. The label **UART-RX-ISR** indicates the entry point. For more information on how to write assembler definitions see the assembler chapter.

```
ASMCODE
l: UART-RX-ISR
  d0 -(a7) l. move,                \ save registers used
  # 0 d0 l. moveq,                 \ clear receivE register
  $l URB MBASE + d0 b. move,       \ fetch it
  d0 $l rx-char l. move,           \ store it
  # -1 $l rx-avail l. move,        \ note char is available
  (a7)+ d0 l. move,                \ restore registers used
  rte,                             \ return
END-CODE

uart-rx-isr  start-vector UARTvector 4 * +  !
```

Setting the interrupt

The interrupt code above can be set to a vector by entering in your source code,

```
uart-rx-isr  start-vector UARTvector 4 * +  !
```

where **START-VECTOR** is the beginning of the vector table and **UARTvector** is the number of the interrupt vector to be set.

Controlling the interrupts

Interrupts can be in one of two states, enabled or disabled.

Enabling interrupts

To enable interrupts use **EI**. Once **EI** has been executed, all interrupts are enabled.

Disabling interrupts

To disable interrupts use **DI**. Once **DI** has been executed, all interrupts are disabled.

A simple example

The following example patches a high-level interrupt service routine (ISR) onto an external timer interrupt. It assumes that the external timer is attached to the level 2 interrupt on the 68000.

The timer ISR

The example ISR increments a variable that can be fetched in the foreground to detect that the timer is working. The ISR also needs to acknowledge that the interrupt has happened.

```
HEX
: TIMER-INIT          \ -- ;
  \ initialise  timer
;

VARIABLE TICKS

 : TIMER-SERVICE      \ -- ; high-level service routine
  1 TICKS +!                \ count 1 sec "ticks"
  \ reset timer
;
```

Patching your ISR onto the timer

The ISR needs to be patched onto the interrupt, and a new entry point generated. Because of the number of available vectors, it is impractical to provide code for every one, although code for the common vectors (including the level 2 autovector) is provided.

```
\ The following adds a new entry point and service word
\ for the level 2 interrupt
\ This is taken from the code provided in INT68K.FTH

DEFER AVEC2-ISR       \ Level 2 interrupt service word
  ASSIGN TIMER-SERVICE TO-DO AVEC2-ISR
```

```
L: AVEC2-INT
 ] AVEC2-ISR RETI [          \ XT of action word
                             \ followed by RETI action

ASMCODE
L: AVEC2-INT0                \ 68000 vector should point here
  { d0-7 a0-6 } -(a7) l. movem,     \ save cpu registers
  # AVEC2-INT a2 l. movea,   \ set up Forth IP
  NEXT-INT w. bra            \ point to int, and go
END-CODE

DECIMAL
AVEC2-INT0 start-vector 26 4* + !   \ patch vector
```

Once this is done, the timer is ready to go, and must now be initialised.

Initialising the timer

```
\ toc4-enable toc4-disable
\ These illustrate the use of save-int and restore-int to
\ alter the interrupt priority in the 68000 SR.

HEX

: TIMER-ENABLE       \ -- ; disable priority 1 interrupts
  SAVE-INT
  $0700 NOT AND              \ Clear int mask bits
  $0100 OR                   \ set new mask
  RESTORE-INT                \ set priority = 1
;

: TIMER-DISABLE      \ -- ; disable priority 2
                            \ and lower interrupts
  SAVE-INT
  0700 NOT AND               \ clear interrupt mask bits
  0200 OR                    \ mask interrupt level 2
  RESTORE-INT                \ set priority = 2
;
```

The timer can be initialised by typing **`TIMER-INIT`** followed by **`TIMER-ENABLE`**.

Testing that the timer is running

The timer can be tested by checking the variable **`TICKS`**:

```
TICKS ?
```

This displays the current value of **`TICKS`**.

Interrupt handlers in detail

When an interrupt occurs and is accepted, a 68000 processor branches through a location (a `vector') in low memory. A conventional interrupt handler written in assembler simply saves any registers and fixed locations it will use, then calls or executes the required routine, and then restores the previous state of registers, locations, and returns. The Forth interrupt handlers work in the same way, with a few additions.

Interrupt handlers written entirely in assembler are written in the usual way. When a Forth **VARIABLE** is referred to by name inside a code section, its data address in RAM is returned. Thus variables can be shared between high level routines and subroutines or assembler interrupt handlers.

The code for the interrupt handler can be found in the file INT68K.FTH in the ROM subdirectory. This source code can be modified and updated as required. The interrupt handler code can be adapted for single-chip use.

Interrupt structure

A separate user area is reserved for the interrupt handler, and is shared by all the high-level interrupt handlers. Consequently high-level Forth interrupts cannot be nested if user variables are changed, although an interrupt that does not use the Forth interrupt handler can be nested. The layout of the interrupt RAM page is identical to that for a task page (see later).

When an interrupt that has a high-level handler occurs, all registers are pushed onto the stack. These are then loaded with the required values for interrupt processing.

The interrupt code causes the processor to jump to an area of memory that contains calls to two words. The first is a call to a word to run the interrupt; this is a **DEFER**red word whose action can be reassigned. The second is a call to the return from interrupt word.

Setting an interrupt

To set the action of an interrupt, simply assign that word to do the appropriate deferred word. The return from interrupt is provided by the second word of the table entry. This means that the action word need contain no return action, and consequently it may be tested from the keyboard. In the earlier example, the variable **TICKS** can be inspected to see that the timer is running. Changes to this structure can be made by rebuilding the Forth kernel with the cross compiler.

In particular, some interrupts may need to be serviced by an assembler routine, and the overhead of the Forth dispatcher is unwanted. In this instance edit the interrupt entry code to call a different routine.

Since 68000 interrupts (or "exceptions") are vectored, the final requirement is to set the vector itself. The 68000 has 256 interrupt vectors. The supplied code provides eighteen interrupt entry points that are predefined for the 68000 CPU.

If you wish to add vectors for processing other exceptions, you should use the code provided in INT68K.FTH as a prototype, and follow a similar structure.

The words **ASSIGN** and **TO-DO** can be used to assign a Forth word to be executed by the interrupt handler in the following manner.

```
ASSIGN action TO-DO xxxx-ISR
```

where “xxxx” is the name of the interrupt, as given in the table below, and action is the Forth word that the interrupt handler must execute. Thus to set up a complete interrupt system the vector must be assigned, and the word connected to the interrupt structure.

```
ASSIGN action TO-DO xxxx-ISR
<<xxxx-INT0>>  start-vector <<vector#>> 4* + !
```

where **<<vector#>>** is the vector number in low memory, as given in the table below. Note that interrupt handlers written in assembler do not need to use these structures at all. They must still, however, set the CPU vectors in low memory.

On some 68000 derivatives there are on-chip peripherals can be programmed to use one or more of the 192 user-defined interrupt vectors. To set up an interrupt system for one of these vectors, an entry point must be written in assembler, a service word must be defined, and a high-level entry point created with the address of the service word and the address of the end-of-interrupt word. (The source code in INT68K.FTH can be used as a prototype for this.) Then the address of the assembler entry point can be installed in the 68000 interrupt vector. If the service word is a **DEFER**red word, a Forth word must be **ASSIGN**ed to it.

Source	Vector #	Entry point	Deferred word
Bus Error	0002	BUSERR-INT0	BUSERR-ISR
Address Error	0003	ADDRERR-INT0	ADDRERR-ISR
Illegal Instruction	0004	ILLOP-INT0	ILLOP-ISR
Division by Zero	0005	DIVZERO-INT0	DIVZERO-ISR
CHK Instruction	0006	CHKINST-INT0	CHKINST-ISR
TRAPV Instruction	0007	TRAPVINST-INT0	TRAPVINST-ISR
Privilege Violation	0008	PRIVVIOL-INT0	PRIVVIOL-ISR
Trace	0009	TRACE-INT0	TRACE-ISR
Opcode A Emulation	0010	OPCODE1010-INT0	OPCODE1010-ISR
Opcode F Emulation	0011	OPCODE1111-INT0	OPCODE1111-ISR
Spurious Interrupt	0024	PHANTOM-INT0	PHANTOM-ISR
Level 1 Autovector	0025	AVEC1-INT0	AVEC1-ISR
Level 2 Autovector	0026	AVEC2-INT0	AVEC2-ISR
Level 3 Autovector	0027	AVEC3-INT0	AVEC3-ISR
Level 4 Autovector	0028	AVEC4-INT0	AVEC4-ISR
Level 5 Autovector	0029	AVEC5-INT0	AVEC5-ISR
Level 6 Autovector	0030	AVEC6-INT0	AVEC6-ISR
Level 7 Autovector	0031	AVEC7-INT0	AVEC7-ISR

Table 6: 68000 interrupts with predefined handlers

Interrupt protection

The word **SAVE-INT** saves the current interrupt status, and disables interrupts, which can then be restored by **RESTORE-INT** which restores the interrupt mask to the state saved by **SAVE-INT**. If simple enabling and disabling of interrupts is required, the words **EI** and **DI** respectively perform these functions. **SAVE-INT** and **RESTORE-INT** are particularly necessary for critical sections of code, and for implementing semaphores.

The word **[I** saves the current interrupt status on the return stack, and disables interrupts, which can then be restored by **I]** which restores the interrupt mask to the state saved by **[I**. If simple enabling and disabling of interrupts is required, the words **EI** and **DI** respectively perform these functions. **[I** and **I]** are particularly necessary for critical sections of code, and for implementing semaphores. In general the use of **[I … I]** requires less code than the use of **SAVE-INT … RESTORE-INT**.

On the 68xxx compiler these words are implemented by code generators, and so it is not necessary to supply target definitions for them. If they are required interactively, example definitions can be found in INT68K.FTH and moved to LIB68K.FTH, and will be included automatically if required, providing that you use the relevant **LIBRARIES … END-LIBS** sequence.

End of interrupt requirements

Certain interrupts on the 68000 require that the programmer explicitly clear the interrupt source before returning from the interrupt. This action must be performed by the high-level interrupt routine. In many cases, resetting the appropriate bit in a status register clears the interrupt source.

Glossary

This glossary contains details of the major words in the interrupt system. Other words exist, but are only used as fractions of the words below. The source code for all these words may be found in INT68K.FTH.

DI --
"d-i"

Disables interrupts.

EI --
"e-i"

Enables interrupts.

RESTORE-INT n –
"restore-int"

Restore the interrupt enable state previously saved by **SAVE-INT**.

SAVE-INT -- n
"save-int"

Saves the current state of the interrupt enable on the stack, and disables interrupts. See **RESTORE-INT**.

[I R: -- x ; save interrupt staus, disable inteerupts
“bracket-i”

Save the current interrupt status on the return stack and disable interrupts. This word can only be used inside a colon definition and **[I** and **I]** must be used in matching pairs.

I] R: ccr -- ; restore CCR from return stack
“i-bracket”

Restore the interrupt status from the return stack. This word can only be used inside a colon definition and **[I** and **I]** must be used in matching pairs.

6 Index