2.1 AS68 ASSEMBLER OPERATION

The GEM DOS Assembler, AS68, assembles an assembly language source program for execution on the 68000 microprocessor. It produces a relocatable object file and, optionally, a listing. You can find a summary of the AS68 instruction set at the end of this section. Exceptions and additions to the standard Motorola instruction set appear in sections 2.6 and 2.7.

2.2 INITIALIZING AS68

If the file AS68SYM.DAT is not on your disk, you must create this file to initialize AS68 before you can use AS68 to assemble files. To initialize AS68, specify the AS68 command, the -I option, and the filename AS68INIT as shown below.

```
{a}AS68 -I AS68INIT
```

AS68 creates the output file AS68SYMB.DAT, which AS68 requires when it assembles programs. After you create this file, you need not specify this command line again unless you reconfigure your system to have different TPA boundaries.

2.3 INVOKING THE ASSEMBLER (AS68)

Invoke AS68 by entering a command with the following form:

```
AS68 [-F pathname] [-P] [-S pathname] [-U] [-L] [-N] [-I]
     [-O object filename] source pathname [>listing pathname]
```

Table 2-1 lists and describes the AS68 command line options.

Table 2-1. Assembler Options

Option	Meaning
-F pathname	Specifies the directory in which the temporary files are created. If this option is not specified, AS68 creates the temporary files in the current default directory.
-I	Initializes the assembler. See Section 2.2 for details.

-P

If specificed, AS68 produces and prints a listing on the standard output device, which, by default, is the console. Redirect the listing, including error messages, to a file with the >listing filename parameter. Note that error messages are produced whether or not the -P option is specified. No listing is produced, however, unless you specify the -P option.

-S pathname

Indicates the directory that contains the assembler initialization file, AS68SYMB.DAT. This file is created when you initialize AS68. AS68 reads the file AS68SYMB.DAT before it assembles a source file. If you do not specify this option, AS68 assumes the initialization file is located in the current default directory.

-U

Causes all undefined symbols in the assembly to be treated as global references.

-L

Ensures all address constants are generated as longwords. Use the -L option for programs that require more than 64K for execution or if the TPA is not contained in the first 64K bytes of memory. If -L is not specified, the program is assembled to run in the first 64K bytes of memory. If an address in the assembly does not fit within one word, an error occurs.

-N

Disables optimization of branches on forward references. Normally, AS68 uses the 2-byte form of the conditional branch and the 4-byte BSR instruction wherever possible (instead of the 6-byte JSR instruction) to speed program execution and reduce instruction size.

-T

Enables AS68 to accept the 68010 microprocessor opcodes.

source filename

Specifies the file to assemble; you must supply this parameter.

>listing filename

Sends a program listing to the standard output device. Use the greater-than symbol, >, to direct the listing to a disk file. The listing includes assembler error messages. Note that if you do not specify -P with a listing filename, only the error messages are redirected to the listing file.

2.4 ASSEMBLY LANGUAGE DIRECTIVES

Table 2-2 lists the AS68 directives.

Table 2-2. AS68 Directives

Directive Meaning

.comm label, expression

The comm (common) directive specifies a label and the size of a common area that programs assembled separately can share. The largest common area of a group with the same label determines the final size of the program common area.

data

The data directive instructs AS68 to change the assembler base segment to the data segment.

bss

The bss (block storage segment) directive instructs AS68 to change the assembler base segment to the block storage segment. You cannot assemble instructions and data in the bss. However, you can define symbols and reserve storage in the bss with the ds directive.

dc operand[,operand,...]

The dc (define constant) directive defines one or more constants in memory. The operands can be symbols or expressions assigned numeric values by AS68, or explicit numeric constants in decimal or hexadecimal, or strings of ASCII characters. You must separate operands with commas. You must enclose string constants in single quotation marks. Each ASCII character is assigned a full byte of memory. The eighth bit is always 0.

You can specify the length of each constant with a single letter parameter (byte = b, word = w, longword = l). You must separate the letter from the dc with a period as shown in the following explanations.

.dc.b

The constants are byte constants. If you specify an odd number of bytes, AS68 fills the odd byte on the right with zeros unless the next statement is another dc.b directive. When the next statement is a dc.b directive, the dc.b uses the odd byte. Byte constants are not relocatable.

.dc.w

The constants are word constants. If you specify an odd number of bytes, AS68 fills the last word on the right with zeros to force an even byte count. The only way to specify an odd number of bytes is with an ASCII constant. Word constants can be relocated.

.dc.l

The constants are longword constants. If less than a multiple of four bytes is entered, AS68 fills the last longword on the right with zeros to force a multiple of four bytes. Longword constants can be relocated.

.ds operand

The define storage directive (ds) reserves memory locations. The contents of the memory that it reserves is not initialized. The operand specifies the number of bytes, words, or longwords that this directive reserves. The notation for these size specifications is shown below.

.ds.b	reserves memory locations in bytes
.ds.w	reserves memory locations in words
.ds.l	reserves memory locations in longwords

.end

The end directive informs AS68 that no more source code follows this directive. Code, comments, or multiple carriage returns cannot follow this directive.

.endc

The endc directive denotes the end of the code that is conditionally assembled. It is used with other directives that conditionally assemble code.

.equ (or =) expression

The equate directive (equ or =) assigns the value of the expression in the operand field to the symbol in the label field that precedes the directive. The syntax for the equate directive are:

label .equ expression
label = expression

The label and operand fields are required. The label must be unique; it cannot be defined anywhere else in the program. The expression cannot include an undefined symbol or one that is defined following the expression. Forward references to symbols are not allowed for this directive.

.even

The even directive increments the location counter to force an even boundary. For example, if specified when the location counter is odd, the location counter is incremented by one so that the next instruction or data field begins on an even boundary in memory.

.globl label[,label...]
.xdef label[,label...]
.xref label[,label...]

These directives make the label(s) external. If the labels are defined in the current assembly, this statement makes them available to other routines during a load by ALN. If the labels are not defined in the current assembly, they become unresolved external references, which ALN links to external values with the same label in other routines. If you specify the -u option, the assembler makes all undefined labels external.

.ifeq expression **.ifne** expression
.ifle expression **.iflt** expression
.ifge expression **.ifgt** expression

These directives test an expression against zero for a specified condition. If the expression is true, the code following is assembled; if false, the code is ignored until an end conditional directive (endc) is found. The directives and the conditions they test are:

.ifeq	equal to zero	.ifle	less than or equal to zero
.iflt	less than zero	.ifne	not equal to zero
.ifgt	greater than zero	.ifge	greater or equal to zero

.ifc 'string1', 'string2'
.ifnc 'string1', 'string2'

The conditional string directive compares two strings. The 'c' condition is true if the strings are exactly the same. The 'nc' condition is true if they do not match.

.offset expression

The offset directive creates a dummy storage section by defining a table of offsets with the define storage directive (ds). The storage definitions are not passed to the linker. The offset table begins at the address specified in the expression. Symbols defined in the offset table are internally maintained. No instructions or code-generating directives, except the equate (equ) and register mask (reg) directives, can be used in the table. The offset directive is terminated by one of the following directives:

bss
data
end
section
text

.org expression

The absolute origin directive (org) sets the location counter to the value of the expression. Subsequent statements are assigned absolute memory locations with the new value of the location counter. The expression cannot contain any forward, undefined, or external references.

.page

The page directive causes a page break which forces text to print on the top of the next page. It does not require an operand or a label and it does not generate machine code.

The page directive allows you to set the page length for a listing of code. If you use this directive and print the source code by specifying the -P option in the AS68 command line, pages break at predefined rather than random places. The page directive does not appear on the printed program listing.

.reg reglist

The register mask directive builds a register mask that can be used by a movem instruction. (See Table 1-1.) One or more registers can be listed in ascending order in the format:

R?[-R[/R?[-R?...]...]]

Replace the R in the above format with a register reference. Any of the following mnemonics are valid:

A0-A7
D0-D7
R0-R15

The following example illustrates a sample register list.

A2-A4/A7/D1/D3-D5

You can also use commas to separate registers as follows:

A1,A2,D5,D7

.section #

The section directive defines a base segment. The sections can be numbered from 0 to 15 inclusive. Section 14 always maps to data. Section 15 is bss. All other section numbers denote text sections.

.text

The text directive instructs AS68 to change the assembler base segment to the text segment. Each assembly of a program begins with the first word in the text segment.

2.5 SAMPLE COMMANDS INVOKING AS68

```
{a}as68 -u -l test.s
```

This command assembles the source file TEST.S and produces the object file TEST.O. Error messages appear on the screen. Any undefined symbols are treated as global

```
{a}as68 -p  smpl.s > smpl.l
```

This command assembles the source file SMPL.S and produces the object file SMPL.O. The program must run in the first 64K of memory; that is, no address can be larger than 16 bits. Error messages and the listing are directed to the file SMPL.L.

2.6 ASSEMBLY LANGUAGE DIFFERENCES

The syntax differences between the AS68 assembly language and Motorola's assembly language are described in the following list.

- In AS68, all assembler directives are optionally preceded by a period (.). For example,

  .equ or equ
  .ds or ds

- AS68 does not support, but accepts and ignores the following Motorola directives:

  comline
  mask2
  idnt
  opt

- The Motorola .set directive is implemented as the equate directive (equ).

- AS68 accepts upper- and lowercase characters. You can specify instructions and directives in either case. However, labels and variables are case-sensitive. For example, the label START and Start are not equivalent.

- For AS68, all labels must terminate with a colon (:). For example,

  A: FOO:

However, if a label begins in column 1, it need not terminate with a colon. If a label is placed as the last statement of your assembly, it must generate code, ie. conditional statements may cause problems but a no op (nop) will be OK.

- For AS68, ASCII string constants can be enclosed in either single or double quotes. For example,

  'ABCD' "ac14"

- For AS68, registers can be referenced with the following mnemonics:

  r0-r15
  R0-R15
  d0-d7
  D0-D7
  a0-a7
  A0-A7

Upper- and lowercase references are equivalent. Registers R0-R7 are the same as D0-D7 and R8-R15 are the same as A0-A7.

- Use caution when manipulating the location counter forward in AS68. An expression can move the counter forward only. The unused space is filled with zeros in the text or data segments.

- For AS68, comment lines can begin with an asterisk followed by an equals sign (* =), but only if one or more spaces exist between the asterisk and the equals sign as follows:

  * = This command loads R1 with zeros.
  * = Branch to subroutine XYZ.

Be sure to include a space after the asterisk, as the location counter is manipulated with a statement of the form:

  *=expr

- For AS68, the syntax for short form branches is bxx.b rather than bxx.s

- The Motorola assembler supports a programming model in which a program consists of a maximum of 16 separately relocatable sections and an optional absolute section. The AS68 distributed with GEMDOS does not support this model. Instead, AS68 supports a model in which a program contains three segments, text, data, and bss as described in the Atari GEMDOS manual.

2.7 ASSEMBLY LANGUAGE EXTENSIONS

The following enhancements have been added to AS68 to make the assembly language more efficient:

- When the instructions add, sub, and cmp are used with an address register in the source or destination, they generate adda, suba, and cmpa. When the clr instruction is used with an address register (Ax),it generates sub Ax, Ax.

- add, and, cmp, eor, or, sub are allowed with immediate first operands and generate addi, andi, cmpi, eori, ori, and subi instructions if the second operand is not register-direct.

- All branch instructions generate short relative branches where possible, including forward references.

- Any shift instruction with no shift count specified assumes a shift count of one. For example, asl rl is equivalent to asl #1,rl.

- A jsr instruction is changed to a bsr instruction if the resulting bsr instruction is shorter than the jsr instruction.

- The .text directive causes the assembler to begin assembling instructions in the text segment. The .data directive causes the assembler to begin assembling initialized data in the data segment.

- The .bss directive instructs the assembler to begin defining storage in the bss. No instructions or constants can be placed in the bss because the bss is for uninitialized data only. However, the .ds directives can be used to define storage locations, and the location counter (*) can be incremented.

- The .globl directive in the form:

    globl label[,label] ...

makes the labels external. If they are otherwise defined (by assignment or appearance as a label), they act within the assembly exactly as if the .globl directive were not given. However, when linking this program with other programs, these symbols are available to other programs. Conversely, if the given symbols are not defined within the current assembly, the linker can combine the output of this assembly with that of others which define the symbols.

- The common directive (comm) defines a common region, which can be accessed by programs that are assembled separately. The syntax for the common directive is:

    comm label, expression

The expression specifies the number of bytes allocated in the common region. If several programs specify the same label for a common region, the size of the region is determined by the value of the largest expression.

The common directive assumes the label is an undefined external symbol in the current assembly.

- The .even directive causes the location counter (*), if positioned at an odd address, to be advanced by one byte so the next statement is assembled at an even address.

- The instructions move, add, and sub, specified with an immediate first operand and a data (D) register as the destination, generate Quick instructions, where possible.

2.8. AS68 ERROR MESSAGES

The GEM DOS assembler, AS68, returns both nonfatal, diagnostic error messages and fatal error messages. Fatal errors stop the assembly of your program. There are two types of fatal errors: user-recoverable fatal errors and fatal errors in the internal logic of AS68.

2.8.1. AS68 Diagnostic Error Messages

Diagnostic messages report errors in the syntax and context of the program being assembled without interrupting assembly. Refer to the Motorola 16-Bit Microprocessor User's Manual for a full discussion of the assembly language syntax.

Diagnostic error messages appear in the following format:

& line no. error message text

The ampersand (&) indicates that the message comes from AS68. The line no. indicates the line in the source code where the error occurred. The error message text describes the error. Diagnostic error messages appear at the console after assembly, followed by a message indicating the total number of errors. In a print-out, they print on the line preceding the error. Table A-1 lists the AS68 diagnostic error messages in alphabetical order.

Table 2-3. AS68 Diagnostic Error Messages

Message Meaning

& line no. backward assignment to *

The assignment statement in the line indicated illegally assigns the location counter (*) backward. Change the location counter to a forward assignment and reassemble the source file.

& line no. bad use of symbol

A symbol in the source line indicated has been defined as both global and common. A symbol can be either global or common, but not both. Delete one of the directives and reassemble the source file.

& line no. constant required

An expression on the line indicated requires a constant. Supply a constant and reassemble the source file.

& line no. end statement not at end of source

The end statement must be at the end of the source code. The end statement cannot be followed by a comment or more than one carriage return. Place the end statement at the end of the source code, followed only by a single carriage return, and reassemble the source file.

& line no. illegal addressing mode

The instruction on the line indicated has an invalid addressing mode. Provide a valid addressing mode and reassemble the source file.

& line no. illegal constant

The line indicated contains an illegal constant. Supply a valid constant and reassemble the source file.

& line no. illegal expr

The line indicated contains an illegal expression. Correct the expression and reassemble the source file.

& line no. illegal external

The line indicated illegally contains an external reference to an 8-bit quantity. Rewrite the source code to define the reference locally or use a 16-bit reference and reassemble the source file.

& line no. illegal format

An expression or instruction in the line indicated is illegally formatted. Examine the line. Reformat where necessary and reassemble the source file.

& line no. illegal index register

The line indicated contains an invalid index register. Supply a valid register and reassemble the source file.

& line no. illegal relative address

An addressing mode specified is not valid for the instruction in the line indicated. Refer to the Motorola 16-Bit Microprocessor User's Manual for valid register modes for the specified instruction. Rewrite the source code to use a valid mode and reassemble the file.

& line no. illegal shift count

The instruction in the line indicated shifts a quantity more than 31 times. Modify the source code to correct the error and reassemble the source file.

& line no. illegal size

The instruction in the line indicated requires one of the following three size specifications: b (byte), w (word), or l (longword). Supply the correct size specification and reassemble the source file.

& line no. illegal string

The line indicated contains an illegal string. Examine the line. Correct the string and reassemble the source file.

& line no. illegal text delimiter

The text delimiter in the line indicated is in the wrong format. Use single quotes ('text') or double quotes ("text") to delimit the text and reassemble the source file.

& line no. illegal 8-bit displacement

The line indicated illegally contains a displacement larger than 8-bits. Modify the code and reassemble the source file.

& line no. illegal 8-bit immediate

The line indicated illegally contains an immediate operand larger than 8-bits. Use the 16- or 32-bit form of the instruction and reassemble the source file.

& line no. illegal 16-bit displacement

The line indicated illegally contains a displacement larger than 16-bits. Modify the code and reassemble the source file.

& line no. illegal 16-bit immediate

The line indicated illegally contains an immediate operand larger than 16-bits. Use the 32-bit form of the instruction and reassemble the source file.

& line no. invalid data list

One or more entries in the data list in the line indicated is invalid. Examine the line for the invalid entry. Replace it with a valid entry and reassemble the source file.

& line no. invalid first operand

The first operand in an expression in the line indicated is invalid. Supply a valid operand and reassemble the source file.

& line no. invalid instruction length

The instruction in the line indicated requires one of the following three size specifications: b (byte), w (word), or l (longword). Supply the correct size specification and reassemble the source file.

& line no. invalid label

A required operand is not present in the line indicated, or a label reference in the line is not in the correct format. Supply a valid label and reassemble the source file.

& line no. invalid opcode

The opcode in the line indicated is non-existent or invalid. Supply a valid opcode and reassemble the source file.

& line no. invalid second operand

The second operand in an expression in the line indicated is invalid. Supply a valid operand and reassemble the source file.

& line no. label redefined

This message indicates that a label has been defined twice. The second definition occurs in the line indicated. Rewrite the source code to specify a unique label for each definition and reassemble the source file.

& line no. missing)

An expression in the line indicated is missing a right parenthesis. Supply the missing parenthesis and reassemble the source file.

& line no. no label for operand

An operand in the line indicated is missing a label. Supply a label and reassemble the source file.

& line no. opcode redefined

A label in the line indicated has the same mnemonics as a previously specified opcode. Respecify the label so that it does not have the same spelling as the mnemonic for the opcode. Reassemble the source file.

& line no. register required

The instruction in the line indicated requires either a source or destination register. Supply the appropriate register and reassemble the source file.

& line no. relocation error

An expression in the line indicated contains more than one externally defined global symbol. Rewrite the source code. Either make one of the externally defined global symbols a local symbol, or evaluate the expression within the code. Reassemble the source file.

& line no. symbol required

A statement in the line indicated requires a symbol. Supply a valid symbol and reassemble the source file.

& line no. undefined symbol in equate

One of the symbols in the equate directive in the line indicated is undefined. Define the symbol and reassemble the source file.

& line no. undefined symbol

The line indicated contains an undefined symbol that has not been declared global. Either define the symbol within the module or define it as a global symbol and reassemble the source file.

2.8.2. User-recoverable Fatal Error Messages

Table A-2 describes fatal error messages for AS68. When an error occurs because the disk is full, AS68 creates a partial file. Erase the partial file to ensure that you do not try to link it.

Table 2-4. AS68 User-recoverable Fatal Error Messages

& cannot create init: AS68SYMB.DAT

AS68 cannot create the initialization file because the path name is incorrect or the disk to which it was writing the file is full. If you used the -S switch to redirect the symbol table to another disk, check the path name. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reinitialize AS68. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

& expr opstk overflow

An expression in the line indicated contains too many operations for the operations stack. Simplify the expression before you reassemble the source code.

& expr tree overflow

The expression tree does not have space for the number of terms in one of the expressions in the indicated line of source code. Rewrite the expression to use fewer terms before you reassemble the source file.

& I/O error on loader output file

The disk to which AS68 was writing the loader output file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

& I/O write error on it file

The disk to which AS68 was writing the intermediate text file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

& it read error itoffset= no.

The disk to which AS68 was writing the intermediate text file is full. AS68 wrote a partial file. The variable itoffset= no. indicates the first zero-relative byte number not read. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

& Object file write error

The disk to which AS68 was writing the object file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

& overflow of external table

The source code uses too many externally defined global symbols for the size of the external symbol table. Eliminate some externally defined global symbols and reassemble the source file.

& Read Error On Intermediate File: ASXXXXn

The disk to which AS68 was writing the intermediate text file ASXXXX is full. AS68 wrote a partial file. The variable n indicates the drive on which ASXXXX is located. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

& symbol table overflow
The program uses too many symbols for the symbol table. Eliminate some symbols before you reassemble the source code.

& Unable to open file filename

The source filename indicated by the variable filename is invalid or has an invalid path name. Check the path name and the filename. Respecify the command line before you reassemble the source file.

& Unable to open input file

The filename in the command line indicated does not exist or has an invalid path name. Check the path name and the filename. Respecify the command line before you reassemble the source file.

& Unable to open temporary file

You used an invalid path name or the disk to which AS68 was writing is full. Check the path name. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reassemble the source file.

& Unable to read init file: AS68SYMB.DAT

The path name used to specify the initialization file is invalid or the assembler has not been initialized. Check the path name. Respecify the command line before you reassemble the source file. If the assembler has not been initialized, refer to Section 5 for instructions.

& Write error on init file: AS68SYMB.DAT

The disk to which AS68 was writing the initialization file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

& write error on it file

The disk to which AS68 was writing the intermediate text is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk. Erase the partial file that was created on the full disk to ensure that you do not try to link it. Reassemble the source file.

2.8.3. AS68 Internal Logic Error Messages

The following are messages indicating fatal errors in the internal logic of AS68:

& doitrd: buffer botch pitix=nnn itbuf=nnn end=nnn
& doitwr: it buffer botch
& invalid radix in oconst
& i.t. overflow
& it sync error itty=nnn
& seek error on it file
& outword: bad rlflg

2.9 INSTRUCTION SET SUMMARY

This section contains two tables that describe the assembler instruction set distributed with GEMDOS. Table 2-5 summarizes the assembler (AS68) instruction set. Table 2-6 lists variations on the instruction set listed in Table 2-5. For details on specific instructions, refer to Motorola's 16-Bit Microprocessor User's Manual, third edition, MC68000UM(AD3)

Table 2-5. Instruction Set Summary

Instruction	Description
abcd	Add Decimal with Extend
add	Add
and	Logical AND
asl	Arithmetic Shift Left
asr	Arithmetic Shift Right
bcc	Branch Conditionally
bchg	Bit Test and Change
bclr	Bit Test and Clear
bra	Branch Always
bset	Branch Test and Set
bsr	Branch to Subroutine
btst	Bit Test
chk	Check Register Against Bounds
clr	Clear Operand
cmp	Compare
dbcc	Test Condition, Decrement, and Branch
divs	Signed Divide
divu	Unsigned Divide
eor	Exclusive OR
exg	Exchange Registers
ext	Sign Extend
illegal	Illegal Instruction
jmp	Jump
jsr	Jump to Subroutine
lea	Load Effective Address
link	Link Stack
lsl	Logical Shift Left
lsr	Logical Shift Right

move	Move
movem	Move Multiple Registers
movep	Move Peripheral Data
muls	Signed Multiply
mulu	Unsigned Multiply
nbcd	Negate Decimal with Extend
neg	Negate
nop	No Operation
no	One's Complement
or	Logical OR
pea	Push Effective Address
reset	Reset External Devices
rol	Rotate Left without Extend
ror	Rotate Right without Extend
roxl	Rotate Left with Extend
roxr	Rotate Right with Extend
rte	Return From Exception
rtr	Return and Restore
rts	Return from Subroutine
sbcd	Subtract Decimal with Extend
scc	Set Conditional
stop	Stop
sub	Subtract
swap	Swap Data Register Halves
tas	Test and Set Operand
trap	Trap
trapv	Trap on Overflow
tst	Test
unlk	Unlink

Table 2-5. Variations of Instruction Types

Instruction	Variation	
add	add	Add
	adda	Add Address
	addq	Add Quick
	addi	Add Immediate
	addx	Add with Extend
and	and	Logical AND
	andi	AND Immediate
	andi	to ccr
	andi	to sr
cmp	cmp	Compare
	cmpa	Compare Address
	cmpm	Compare Memory
	cmpi	Compare Immediate
eor	eor	Exclusive OR
	eori	Exclusive OR Immediate
	eori to ccr	
	eori to sr	
move	move	Move
	movea	Move Address
	moveq	Move Quick
	move to ccr	
	move to sr	
	move from sr	
	move to usp	
neg	neg	Negate
	negx	Negate with Extend
or	or	Logical OR
	ori	OR Immediate
	ori to ccr	
	ori to sr	OR Immediate to Status Register
sub	sub	Subtract
	suba	Subtract Address
	subi	Subtract Immediate
	subq	Subtract Quick
	subx	Subtract with Extend