Motorola Microcontroller Division
6501 William Cannon Drive West, Austin, TX 78735
Phone: (512) 895-7634, FAX: (512) 895-1902

Document Number: 1001
Revision: 1.0
Date: 26 October, 1999

MOTOROLA STANDARDS
Document #1001, Version 1.0
SPECIFICATION FOR
Motorola 8- and 16-Bit ASSEMBLY LANGUAGE INPUT STANDARD

PROPOSAL:

This specification proposes to define an assembly language input standard for Motorola 8- and 16-bit microcontrollers. In essence, this is a framework so that any compliant assembly language tool that generates MCU machine code can assemble either compiler-generated code or human developed code from any other compliant tool.

BACKGROUND:

The need for this specification arose from a meeting held at the November, 1998 Embedded Systems Conference where a number of third-party suppliers asked Motorola to develop an assembly language input standard. Currently each supplier follows a structure that each has developed independent of the others.

IMPACT:

Given the number of years Motorola MCUs have been on the market and the number of assemblers that are also on the market, changing the assemblers to meet the standard may incur some engineering costs that the market may not bear. The technical impact may not be that great but this depends on how close a supplier's assembler is to the proposed specification.

Motorola Microcontroller Division
6501 William Cannon Drive West, Austin, TX 78735
Phone: (512) 895-7634, FAX: (512) 895-1902

Document Number: 1001
Revision: 1.0
Date: 26 October, 1999

Document #1001
Version 1.0

SPECIFICATION FOR
Motorola 8- and 16-Bit ASSEMBLY LANGUAGE INPUT STANDARD

26 October, 1999

Motorola Microcontroller Division
6501 William Cannon Drive West, Austin, TX 78735
Phone: (512) 895-7634, FAX: (512) 895-1902

Document Number: 1001
Revision: 1.0
Date: 26 October, 1999

TABLE of CONTENTS

1. PURPOSE

This specification defines a standard source code format for assemblers of Motorola 8-bit and 16-bit microcontroller machine code. See the appendixes for the Motorola MCUs covered by this specification. This source code format is intended for use with either human developed or compiler generated assembly source code.

Motorola microcontroller software developers using this standard shall be able to substitute compliant assemblers without modifying their code. This specification encompasses a complete set of features necessary for a macro assembler. The assembly code shall, by default, be case sensitive except where indicated since C compilers generate case-sensitive code.

2. SCOPE

This standard is a generalized format for assembly language source code. This standard is a basic format, which encompasses the major functions of a macro assembler but does not prevent individual suppliers from including enhanced features to suit unique customer needs. All code written according to this standard shall assemble on any compliant assembler but all code that is written for any given assembler need not comply with the standard.

3. LIMITATIONS

This standard applies to Motorola's 8- and 16-bit microcontrollers See the appendixes for the MCUs covered by this specification. It is intended that both software suppliers and developers use this standard. Motorola invites suppliers to use the standard as a set of minimum requirements for their assemblers. Motorola recommends that developers use this standard as a guideline for writing code, which can be easily ported from one assembler to another.

3.1. EXTENSIONS

Suppliers may augment their assemblers with extensions. Extensions may require additional directives or features not defined in this standard. These extensions may or may not be portable.

4. REFERENCED DOCUMENTS

[1] M68HC05 Applications Guide, M68HC05AG/AD 3.0, Motorola

[2] 68CPU08 Reference Manual, CPU08RM/AD 2.0, Motorola

[3] M68HC11 Reference Manual, M68HC11RM/D 3.0, Motorola

[4] 68HC12 CPU12 Reference Manual, CPU12RM/AD 1.0, Motorola

[5] 68CPU16 Reference Manual, CPU16RM/AD 1.0, Motorola

[6] ISO/IEC 14977: 1996(E)

5. TERMINOLOGY

5.1. ASCII - American Standard Code for Information Interchange

5.2. ABSOLUTE - a value independent of the beginning of a code or data section.

5.3. ABSOLUTE ASSEMBLER – an assembler tool which produces machine code directly without the step of resolving instruction address by a Linker tool.

5.4. BIT - a single piece of binary information. Contains the values 0 or 1.

5.5. BYTE – eight (8) bits.

5.6. CARRIAGE RETURN - ASCII code 13 ($0d).

5.7. DATA - binary information for a program generated from assembler directives.

5.8. DIRECTIVE – is a command placed in the assembly source file to tell the assembler to take an action based on the content of the command.

5.9. EOL - end of line, in source files, a character or a sequence of characters that defines the standard end-of-line sequence for the local operating system. Depending on the

system, the end-of-line sequence can be a return, a line feed, or return and a line feed.

5.10. INPUT STREAM - source code with include files inserted, macros expanded, and conditional assembly paths determined.

5.11. INSTRUCTION - a program statement that specifies an operation to be performed by the microcontroller, along with the values or locations of operands. This statement represents the programmer's request to the MCU to perform a specific operation.

5.12. LINE FEED - ASCII code 10 ($0a).

5.13. LINKER – is a tool that accepts relocatable object files from a relocatable assembler. This tool usually combines relocatable objects and resolves machine instruction addresses into actual addresses to create the final binary executable code.

5.14. LONG WORD – four (4) bytes.

5.15. MACHINE INSTRUCTION. - A binary number that directs the operation of a microcontroller. Compilers and assemblers convert source instructions to machine instructions.

5.16. MACHINE CODE – is machine instructions and associated data contained in the microcontroller memory.

5.17. MODULE - a single input stream.

5.18. MCU – microcontroller unit. It is comprised of a microprocessor, memory, and I/O. It is usually considered a self-contained controller.

5.19. NEWLINE - 1. [tech-speak, primarily UNIX] the ASCII LF character (0001010), used under UNIX as a text line terminator. A bell-labs-ism rather than a Berkeley-ism; interestingly (and unusually for UNIX jargon), it is said to have originally been an IBM usage. (Though the term `newline' appears in ASCII standards, it never caught on in the general computing world before UNIX). 2. More generally, any magic character, character sequence, or operation (like Pascal's writeln procedure) required to terminate a

text record or separate lines.

5.20. PATH - sequence of valid directory names.

5.21. PATHNAME - sequence of valid directory names followed by a filename.

5.22. PORT - to make source code written for one assembler readable by another.

5.23. RELOCATABLE - all values are relative to the beginning of a code or data section.

5.24. RELOCATABLE ASSEMBLER – an assembler tool, which produces machine code that must be linked to other machine code to resolve actual instruction, addresses.

5.25. SECTION - is an atomic contiguous unit representing a memory region in the Motorola microcontroller architecture.

5.26. SOURCE CODE - ASCII text which is readable by an assembler.

5.27. SOURCE FILE - the file containing the source code for an assembler.

5.28. WHITESPACE - one or more blanks (ASCII code 32) ($20) or horizontal tabs (ASCII code 9) ($09) in any order used to separate symbols, and to make programs easier for humans to read.

5.29. WORD – two (2) bytes.

6. CONVENTIONS

The syntax of the assembler is defined using Backus-Naur Form (BNF). The Backus-Naur Form (BNF) is a convenient and widely accepted means for writing down the grammar of a context-free language. Context-free grammars are frequently used to specify programming languages, operating system commands, and other types of computer input. Numerous variants of BNF have been introduced and practically every compiler-design textbook and programming-language standard defines its own version.

Although there are many variants and extensions of BNF, it essentially consists of the following notation:

::= the production symbol for declaring production rules
| the disjunction symbol for separating alternate production rules
“ double-quotation marks for delimiting literal strings
[ ] square-brackets for delimiting optional terms
{} curly brackets surround items that can repeat zero or more times
( ) parentheses for grouping terms

6.1. TERMINAL EXPRESSIONS

Terminal expressions are represented in **bold** typeface. The following are some terminals that are used in this document.

Terminal	*Description*
EOL	The end of line.
ASCII Text	Any sequence of ASCII characters.

6.2. NON-TERMINAL EXPRESSIONS

All non-terminal expressions are represented in *Italics.*

6.3. OPERATORS

The table below summarizes the EBNF operators used in the grammar throughout this standard.

Expression	*Description*
expr ::= expr1	Assignment, replace expr with expr1
on | **off**	Alternative, **on** or **off**
(argl | arg2 | arg3)	Select exactly one of the alternatives from within the parentheses
arg [,arg]	Optional, items in the square brackets [ ] are optional
a-z	Range, in the range from **a** to **z**
[**space**]+	Repeat, one or more space

6.4. KEYWORDS

Bold Italics indicates a directive keyword.

7. OVERVIEW

The remainder of this standard describes the language and functional requirements for a compliant assembler. Section 1 describes the purpose of the standard. Section 8 details the assembly language format for the standard. Section 9 outlines the instruction formats used for 8- and 16-Bit Motorola Controllers. The appendixes outline the reserved identifiers used for

Motorola Controllers. Section 10 contains a context-sensitive grammar describing the syntax of the assembly language.

8. LANGUAGE AND FUNCTIONAL REQUIREMENTS

8.1. INTRODUCTION

This standard specifies a set of statement formats, macro formats, section requirements, assembly language directives, and operators, which all compliant assemblers shall recognize.

8.2. INPUTS

This section outlines all the syntax and functionality of the assembler inputs. All compliant assemblers must accept one or more input files. Refer to section 10 for further information about the grammars included in the following sections.

8.2.1. Character Set

The assembly language source file is comprised of a series of ASCII characters.

characters	::=	*character*
	|	*character characters*

- The assemblers shall recognize at least the first 128 ASCII *characters* (decimals 0 -127).

NOTE: It is strongly recommended that each supplier make their assemblers recognize the extended ASCII character set (decimals 0 - 255) to make more code portable.

8.2.2. Assembly Language Statements

An assembly language source file consists of a sequence of assembly language statements, which direct the assembly process.

statement	::=	*comment* **EOL**
	|	[*label*] *comment* **EOL**
	|	[*label* | *whitespace*] *instruction comment* **EOL**
	|	[*label* | *whitespace*] *directive comment* **EOL**
	|	*label labeled_directive comment* **EOL**
	|	[*whitespace*] *unlabeled_directive comment* **EOL**

- An assembly language statement shall consist of a line of text containing an assembler directive, a comment, a label, or a machine instruction statement.
- A line shall be terminated by the system **EOL**.

- An assembly language statement shall reside entirely on one line.
- A line shall be significant to at least 255 characters exclusive of the **EOL**.

8.2.3. Comments

A comment is text that is not assembled. It is merely for annotation of line or section of code for human readability.

comment ::= [*whitespace*] [**;** *text*]
| [*whitespace*]

- A line containing only *whitespace* shall be treated as a comment.
- A line containing only an **EOL** shall be treated as a comment.
- Any text following a (;) shall be treated as a comment.

8.2.4. Reserved Identifiers

All register mnemonics for the target processor shall be reserved. All unlabeled directives are reserved. No other mnemonics shall be reserved. For details about reserved identifiers, see the appendixes.

8.2.5. Location Counter

An asterisk (*) in the operand field of an assembly statement shall refer to the current location counter. This indicates the value of the current memory address at the start of the current instruction line.

location_counter ::= *

- By default, the location counter shall be set to 0 at the beginning of the assembly process.
- In a relocatable assembly, the location counter is set to 0 with respect to that section's starting point.

8.2.6. Sections

A ***section*** is an atomic contiguous unit representing a memory region in the Motorola microcontroller architecture. Every assembly language source file, including those produced by a compiler, defines one or more sections. See section 8.2.13.15 for *BNF*.

- Sections can be anonymous, with only an absolute start address, or can be named using the ***section*** directive.
- Named sections can be absolute, that is, given an absolute start address, or relocatable.
- Relocatable sections shall be given an actual start address at link time.
- Sections are units of relocation.
- Sections in compiler source files shall have the characteristics outlined in this section.

- Sections are only applicable with relocatable assemblers.
- Each file may contain up to 255 sections.
- Sections may be empty.
- The user shall give each named section a *section_name* using the ***section*** directive.
- *section_name*(s) shall be significant up to at least 32 characters.
- Reserved identifiers may not be used as section names.
- *section_name*(s) must be uniquely defined the first time, but may be referenced any number of times elsewhere in the file. Additional references cause a switch to the *named_section*.
- Sections with the same *section_name* shall be concatenated in the order that they are encountered in the input stream.
- The ***section*** directive may create new sections.

8.2.7. Expressions

Expressions are equations consisting of symbols, constants and operators. A compliant assembler is not required to support *complex_relocatable_expression.* However, they are included here to help clarify the difference between the kind of *expression.* If there are *whitespace* characters in expressions they are ignored. The *whitespace* is there only for readability by humans.

expression	::=	*absolute_expression*
	|	*simple_relocatable_expression*
	|	*complex_relocatable_expression*

- An *expression* shall evaluate to at least 32 significant bits.
- Signed arithmetic shall be used.

8.2.7.1. Primary Expression

A *primary_expression* is either a symbol or a constant.

primary_expression	::=	*numeric_constant*
	|	*symbol*

8.2.7.2. Absolute Expressions

An *absolute_expression* is an expression, which is resolved to an absolute value at assembly time.

absolute_expression	::=	*unary _operator absolute _expression*
	|	*absolute_expression binary _operator absolute_expression*
	|	*relocatable_expression - relocatable_expression*
	|	*primary_expression*

- An *absolute_expression* is an expression, which resolves completely to a numeric value at assembly time.
- An *absolute_expression* shall not change at link time.
- *absolute_expression*(s) shall be supported by all compliant assemblers.
- Both sub-expressions of the expression *relocatable_expression - relocatable_expression* must be in the same section of the same source file.

8.2.7.3. Simple Relocatable Expressions

A *simple_relocatable_expression* is an expression, which is resolved at assembly time to an absolute offset from the start of a relocatable section. A *simple_relocatable_expression* is only applicable to relocatable assemblers.

simple_relocatable_expression	::=	*relocatable_expression + absolute_expression*
	|	*absolute_expression + relocatable_expression*
	|	*relocatable_expression - absolute_expression*
	|	*symbol*

- A *relocatable_expression* is an expression, which resolves to an absolute offset from the start of a section at assembly time.
- *simple_relocatable_expression*(s) shall be valid for all compliant relocatable assemblers.
- A *relocatable_expression* shall be updated at link time to reflect the relocated address of the section.

8.2.7.4. Complex Relocatable Expressions

A *complex_relocatable_expression* is an expression, which cannot be resolved to an absolute value or absolute offset from the beginning of a section by the assembler. *complex_relocatable_expression*(s) are any expression, which is neither a simple relocatable_expression nor an absolute_expression.

- *complex_relocatable_expression*(s) are not supported by this standard.

8.2.8. Symbols

A symbol is a tag for an expression. A label is a mechanism for defining symbols. Symbols only have attributes such as their section number or they may be either external or static. Reserved identifiers may not be used as symbols.

8.2.8.1. Labels

Labels are used to define symbols. Statement lines begin with a label field, which the programmer may use to symbolically refer to the memory location of the instruction being assembled. Labels must end with a colon character (the colon does not become part of the label).

Labels may stand alone on a line, followed by a **EOL** or semi-colon comment, or they may lead into the opcode field, separated by one or more tab or space characters. Statement lines, which do not have labels, must use one or more *whitespace* characters to delimit the next field.

label ::= [*whitespace*] *symbol**:*** [*whitespace*]

- Labels shall be significant to at least 32 characters.
- Reserved identifiers may not be used as labels.
- By default, labels shall be case sensitive.
- An assembler shall only accept a *symbol* followed by a colon.

8.2.9. Constants

Constants shall not vary from the point that they are first defined throughout the assembly process.

8.2.9.1. Decimal Constants

A *decimal_constant* is a series of digits from the set **0-9.**

decimal_constant ::= *decimal_digits*
| *decimal_digit decimal_digits*

8.2.9.2. Binary Constants

A *binary_constant* consists of the character **%** followed by a non-empty string of digits from the set **0-1.**

binary_constant ::= *% binary_digits*

8.2.9.3. Octal Constants

An *octal_constant* consists of the character **@** followed by a non-empty string of digits from the set **0-7.**

octal_constant ::= *@ octal_digits*

8.2.9.4. Hexadecimal Constants

A *hex_constant* consists of the character **$** followed by the non-empty string of characters form the set **0-9, a-f,** and **A-F**.

hex_constant ::= *$ hex_digits*

8.2.9.5. String Constants

A *string_constant* is a sequence of zero or more ASCII characters proceeded and followed by a single or double quote.

string_constant ::= *'string'*

| *"string"*

- Double quotes shall be permitted only within strings delimited by single quotes.
- Single quotes shall be permitted only within strings delimited by double quotes.

8.2.9.6. Floating Point Constants

All assemblers need not support floating point constants.

8.2.10. Operators

Compliant source code may contain the following operators:

	Description	***Associativity***
( )	Parenthesis	Right to Left
< > page	Force direct address, index or immediate value to 8-bits Force extended address, index or immediate value to 16-bits Access 8-bit page number from bits 16-23 of 24-bit value	Right to Left Right to Left Right to Left
* / %	Multiplication, Division, Modulus	Left to Right
~ + -	One's Complement, Unary Plus, Unary Minus	Right to Left
+-	Binary Addition, Binary Subtraction	Left to Right
<< >>	Shift Left, Shift Right	Left to Right
== !=	Equal To, Not Equal To	Left to Right
< <=	Less Than, Less Than Equal,	Left to Right
> >=	Greater Than, Greater Than Equal	Left to Right
&	Bitwise And	Left to Right
^	Bitwise Exclusive Or	Left to Right
|	Bitwise Or	Left to Right

8.2.11. Instructions

Instructions for Motorola Microcontrollers shall conform to the formats given in their respective *Reference Manuals* referenced in Section 4. All instruction formats are listed in section 9.

8.2.12. Assembler Directives

Assembler directives are commands, also known as pseudo-operations, that inform the assembler of information about a program rather than directly generating object code. Each assembler as defined below shall recognize the following assembler directives. Some suppliers may give additional capabilities to directives in their assemblers, but only the capabilities described here shall be considered standard.

8.2.12.1. Conditional Directives - *if, else, elseif, endif*

The assembler supports conditional assembly of source statements, allowing sections of code within a source file to be included or excluded when the file is assembled. Assembly-time conditions may be specified through the use of arguments in the case of macros, and also via symbol definitions using the equ and set directives. The conditional directives are ***if***, ***else***, ***elseif***, and ***endif.***

unlabeled_directive	::=	***if*** *whitespace absolute_expression*
	|	***else***
	|	***elseif*** *whitespace absolute_expression*
	|	***endif***
if	:: =	***ifdef*** | ***ifndef***
elseif	::=	***ifdef*** | ***ifndef***

- If *expression* evaluates to true (a non-zero value) all assembly language statements directly following the *expression* until the next matching ***else, elseif*** or ***endif*** (whichever comes first) shall be assembled.
- If *expression* evaluates to false (a zero value) all assembly language statements directly following the *expression* until the next matching ***else, elseif*** or ***endif*** (whichever comes first) shall not be assembled.
- The ***else*** and ***elseif*** directive shall be optional.
- An ***endif*** directive must be used to terminate the conditional.
- The user shall be allowed to nest conditional directives at least 16 levels deep.
- expression must be an absolute_expression.
- *expression* must not contain forward references.
- *expression* may contain any of the operators listed in paragraph 8.2.10.

8.2.12.2. Define Constant – dc

The *define_constant* directive may be used to reserve and initialize constants.

directive	::=	*define_constant*
define_constant	::=	***dc****.size whitespace expression_list*
size	::=	**b** | **w** | **l**

- The ***dc.b*** directive shall reserve one byte per expression at the current location counter and initialize it to expression.
- The ***dc.w*** directive shall reserve one word per expression at the current location counter and initialize it to expression.
- The ***dc.l*** directive shall reserve one long word per expression at the current location counter and initialize it to expression.
- If *expression* is a string constant, one byte should be reserved for each character in the string and initialized to the corresponding character in the string.

- If *expression* is a string constant and the length of the string is greater than or equal to *size*, the string shall be aligned on a *size* boundary and the bytes shall be left justified and null filled from the right.
- No alignment shall be done if *expression* is not a string constant.
- At least 10 *expression* arguments shall be allowed.
- The *symbol* defined by *label* shall be assigned the value of the location counter of the first byte of the first constant.
- *label* must be a valid label.
- If *expression* is numeric and exceeds the value represented in *size* bits, then expression shall be truncated to the least significant *size* bits.
- All symbols in *expression* must be defined.

8.2.12.3. Define Constant *Block – dcb*

The ***dcb***.*size* directive is used to reserve and initialize constant blocks.

directive	::=	*define_constant_block*
define_constant_block	::=	***dcb***.*size whitespace absolute_expression, expression*
size	::=	**b** | **w** | **l**

- The *absolute_expression* is the number of items of *size* to reserve.
- The maximum value for *absolute expression* is $FFFFFFFF.
- The minimum value for *absolute expression* is 1.
- *expression* is the initialization constant.
- There shall be no de facto maximum value for *expression*.
- The ***dcb.b*** directive shall reserve *absolute_expression* bytes at the current location counter.
- The ***dcb.w*** directive shall reserve *absolute_expression* words at the current location counter
- The ***dcb.l*** directive shall reserve *absolute_expression* longwords at the current location counter.
- No alignment shall be done.
- If present, *label* must be a valid label.

8.2.12.4. Define Storage – *ds*

The ***ds***.*size* directive shall be used to reserve data space.

directive	::=	*define_storage*
define_storage	::=	***ds***.*size whitespace expression*
size	::=	**b** | **w** | **l**

- *expression* must be an absolute_expression.
- No symbol in *expression* may be a forward or external reference.
- The minimum value of *expression* shall be one (1).
- The **ds** directive without *.size* shall default to **ds.b**.
- There shall be no de facto maximum value for *expression*, but *expression* must fit into 32 bits.
- The **ds.b** directive shall reserve *expression* bytes at the current location counter.
- The **ds.w** directive shall reserve *expression* words at the current location counter
- The **ds.l** directive shall reserve *expression* long words at the current location counter.
- No alignment shall be done.
- If present, *label* must be a valid label.
- If present, the symbol defined by *label* shall be assigned the value of the location counter of the first byte.

8.2.12.5. Equate Symbol Value – *equ*

The ***equ*** directive shall be used to assign a permanent value to a *label*.

labeled_directive ::= ***equ*** *whitespace expression*

- The ***equ*** directive shall assign the value of *expression* to the *symbol* defined by *label.*
- *expression* must be a valid expression.
- The value assigned by the ***equ*** directive shall be held constant throughout the assembly process.
- *label* shall be mandatory.
- *label* must be a valid label.
- *label* must not be defined more than once.

8.2.12.6. Include File – *include*

The ***include*** directive shall insert another assembly source file into the current input stream.

unlabeled_directive ::= ***include*** *string_constant*

- The ***include*** statement shall be replaced with the contents of the file *string_constant* in the input stream.
- *string_constant* must be a legal pathname for the host operating system.
- *string_constant* shall be enclosed in single or double quotes.
- The user shall be able to nest files at a minimum of 30 deep.

8.2.12.7. Origin – *org*

The ***org*** directive sets the location counter to an offset from the beginning of a section. If ***org*** is not the first directive in a section, it shall create a new section.

unlabeled_directive ::= ***org*** *whitespace expression*

- expression must be an absolute_expression.
- *expression* may not contain any forward or external references.
- The location counter shall be set to *expression*.

8.2.12.8. Define Section – *section*

The ***section*** directive shall define a new section of code and/or data. S*ections* are named.

The second form of the ***section*** directive sets the location counter to the current value of the named section. Before switching sections, the assembler saves the address of the current location counter for a future section invocation of the previously used section. Values used in the ***section*** expression must not contain any forward references.

labeled_directive ::= ***section*** *whitespace [short]*
| ***section*** *whitespace section_name*

- The *section* directive is only applicable with relocatable assemblers.
- At least 255 separate sections may be declared.
- Only one *section_name* may be assigned to a section.
- A *section_name* must be significant (unique) to at least 32 characters.
- Only one section named *section_name* may be defined for each module.
- By default, the location counter shall be set to zero at the beginning of a section.
- All machine instruction statements and data declaration statements following a section directive up to the next *section* shall be placed in the defined section.

8.2.12.9. Set Value of Symbol – *set*

The ***set*** directive may be used to assign or reassign a value to a label.

labeled_directive ::= ***set*** *whitespace expression*

- The ***set*** directive shall assign the value of *expression* to the *symbol* defined by *label.*
- *expression* must be a valid expression.
- *expression* must not contain any forward references.
- *label* may be reassigned by another set directive.
- *label* may not be used in an ***equ*** directive.
- A ***set*** directive may not reassign the value of a label set by another directive.
- If *label* is referenced after it is defined with a ***set*** directive, the most recent ***set*** directive

shall determine the value of label.

8.2.12.10. External Symbol Definition – *xdef*

The *external_definition* directives shall be used to define a symbol, which is available to other modules at link time. This directive is not required for absolute assemblers.

external_definition ::= ***xdef*** *symbol_list*

- *symbol* must be a valid symbol.
- The *external_definition* directives must be able to handle at least 10 symbols.
- *symbol* may be a forward reference.
- *symbol* shall not be allowed in more than one *external_definition* directive.
- *symbol* may not be a macro name.
- *symbol* may not be a section name.
- If a ***set*** directive defines *symbol*, then the value given to *symbol* shall be the value of the last ***set*** directive prior to the ***xdef***.

8.2.12.11. External Symbol Reference – *xref*

The *external_reference* directives shall declare a symbol as being defined in another module, which is available at link time. External symbols declared with these directives shall only have two bytes of space reserved upon each occurrence. Normally the ***xref*** directive defines external reference on a word boundary. The ***xrefb*** directive allows for external definitions on a byte boundary. This directive is not required for absolute assemblers.

external_reference ::= ***xref***
| ***xrefb***

- The *external_reference* directives must be able to handle at least 10 symbols.
- *symbol* may not be previously defined.
- *symbol* may not be defined as a macro.
- All external symbols must be declared in an *external_reference* directive.
- For each *symbol* declared in an *external_reference* directive, the *size boundary* of that directive must match the *size boundary* of the *external_reference* directive in which *symbol* is defined.

9. INSTRUCTION FORMATS

The format for machine instruction mnemonics is outlined below.

9.1. ADDRESSING MODES AND OPERAND FORMS

The operand forms and addressing modes for all of the CPUs detailed in this specification are shown in the following table. Bold characters in the operand syntax column are literal characters. Non-terminal expressions are shown in Italics and explained in the next table.

Operand Form	Address Mode	Description	Operand Syntax
inh	INH	Inherent	No operands
txg	INH	transfer/exchange inherent	*abcdxys*, *abcdxys*
sxg	INH	sign extend inherent	*abc*, *dxys*
imm8	IMM	8-bit immediate	*#oper8i*
imm16	IMM	16-bit immediate	*#operl6i*
Ind0	IDX	Indexed with no extensions... 5-bit constant offset auto pre-increment auto pre-decrement auto post-increment auto post-decrement accumulator offset	 *operx5, xysp* *operx3, +xys* *operx3, -xys* *operx3, xys+* *operx3, xys-* *abd, xysp*
ind1	IDX1	Index register with signed 9-bit offset (1 extension byte)	*operx9, xysp*
ind2	IDX2	Index register with signed 16-bit offset (2 extension bytes)	*operx16, xysp*
indi0	[D,IDX]	Index register d indirect	*[d, xysp]*
indi2	[IDX2]	Index register 16-bit indirect	*{operxl6, xysp}*
rel8	REL	8-bit relative	*rel8*
rel9	REL	9-bit relative	*abdxys, rel9*
rel16	REL	16-BIT RELATIVE	*rell6*
dir	DIR	Direct	*oper8a*
ext	EXT	Extended	*operl6a*
mask8	imm value	Bit mask for bit instructions	*msk8*
page	imm value	8-bit extended memory select	*page*

This table explains the Non-terminal expressions shown in Italics in the previous table.

Non-Terminal Expression	**Expansion**	***expression* Evaluates to...**
abc	(a | b | ccr)	
abcdxys	(a | b | ccr | d | x | y | sp)	
abd	(a | b | d)	
abdxys	(a | b | d | x | y | sp)	
dxys	(d | x | y | sp)	
msk8	expression	An unsigned 8-bit value used as a bit mask
oper8i	expression	8-bit immediate data, signed or unsigned -128 through +255
oper16i	expression	16-bit immediate data, signed or unsigned -32768 through +65535
oper8a	expression	Low 8-bits of a 64k-byte address in the range $0000 through $00FF
oper16a	expression	A 16-bit address in the range $0000 through $FFFF
operx3	expression	An unsigned number in the range 1 through 8
operx5	expression	A signed number in the range -16 through +15
operx9	expression	A signed offset (-256 through +255) used in effective address calculation
operx16	expression	A signed offset (-32768 through +32767) used in effective address calculation
page	expression	An unsigned 8-bit value used as a program page (bank) number 0 through 255
rel8	expression	A signed offset (-128 through +127) from the next address after the offset
rel9	expression	A signed offset (-256 through +255) from the next address after the offset
rel16	expression	A signed offset (-32768 through +32767) from the next address after the offset
xys	(x | y | sp)	
xysp	(x | y | sp | pc)	

Motorola Microcontroller Division
6501 William Cannon Drive West, Austin, TX 78735
Phone: (512) 895-7634, FAX: (512) 895-1902

Document Number: 1001
Revision: 1.0
Date: 26 October, 1999

10.ASSEMBLER SYNTAX GRAMMAR

The Assembler Input Standard shall use a context-sensitive grammar. For example, the position of a character in a line is important, as is whitespace, which is often used as a delimiter.

10.1. SYNTAX VERSUS SEMANTICS

The grammar only addresses syntax and does not resolve semantic issues. For example, the syntax cannot specify the difference between an absolute and a relocatable symbol because this is resolved during lexical analysis and not parsing.

10.2. NOTATION

The grammar is implemented in *Extended Backus Naur Form* (EBNF). The notation used is defined in paragraph 1.5.

10.3. SYSTEM DEPENDENT TERMINAL CHARACTERS

EOL	::=	System dependent end of line character(s)
printable_character	::=	System dependent printable ASCII characters

10.4. GRAMMAR

characters	::=	*character*
	|	*character characters*
statement	::=	*comment* **EOL**
	|	[*label*] *comment* **EOL**
	|	[*label* | *whitespace*] *instruction comment*
	|	[*label* | *whitespace*] *directive comment* **EOL**
	|	*label labeled_directive comment* **EOL**
	|	[*whitespace*] *unlabeled_directive comment* **EOL**
whitespace	::=	[**space** | **tab**]+
comment	::=	[*whitespace*] [**;** *text*]
	|	[*whitespace*]
text	::=	***printable_character***
	|	***printable_character*** *text*
label	::=	[*whitespace*] *symbol*: [*whitespace*]
unlabeled directive	::=	***if*** *whitespace absolute_expression*
	|	***else***
	|	***elseif*** *whitespace absolute_expression*
	|	***endif***
	|	***include*** *whitespace string_constant*

	|	***org*** *whitespace expression*
	|	***ifdef*** *whitespace expression*
	|	***ifndef*** *whitespace expression*
labeled_directive	::=	***equ*** *whitespace expression*
	|	***set*** *whitespace expression*
	|	***section*** *whitespace [section_number | short]*
	|	***section*** *whitespace section_name*
directive	::=	*define_constant whitespace expression_list*
	|	*define_storage whitespace expression*
	|	*external_reference whitespace symbol_list*
	|	*external_definition whitespace symbol_list*
define_constant	::=	***dc****.size whitespace expression_list*
define_constant_block	::=	***dcb.****size whitespace absolute_expression, expression*
define_storage	::=	***ds****.size whitespace expression*
size	::=	**b** | **w** | **l**
argument_list	::=	*argument*
	|	*argument, [whitespace] argument_list*
argument	::=	*expression*
	|	*string_constant*
symbol_list	::=	*symbol*
	|	*symbol, [whitespace] symbol_list*
symbol	::=	*initial_character*
	|	*initial_character characters*
initial_character	::=	**a-z** | **A-Z** | **_**
characters	::=	*character*
	|	*character characters*
character	::=	*initial_character* | **0-9**
base	::=	**0-255**
flag	::=	**on** | **off**
instruction	::=	*mnemonic whitespace operand_list*

NOTE: See **section 9** or the appropriate MCU Reference Manual for a description of the mnemonic.

operand_list	::=	*operand*
	|	*operand, [whitespace] operand_list*
operand	::=	*expression*
	|	*register_mnemonic*
	|	*location_counter*

NOTE: See **the appendixes** or the appropriate MCU Reference Manual for a description of the register-mnemonic.

location_counter	::=	*
expression _list	::=	*expression*
	|	*expression, [whitespace] expression_list*
expression	::=	*absolute-expression*
	|	*simple-relocatable-expression*

	|	*complex-relocatable-expression*
absolute_expression	::=	*unary _operator absolute_expression*
	|	*absolute_expression binary-operator absolute_expression*
	|	*relocatable_expression - relocatable_expression*
	|	*primary-expression*
simple_relocatable_expression	::=	*relocatable_expression + absolute_expression*
	|	*absolute_expression + relocatable_expression*
	|	*relocatable_expression - absolute_expression*
	|	*symbol*
primary_expression	::=	*numeric_constant*
	|	*symbol*
constant	::=	*numeric _constant*
	|	*string_constant*
numeric constant	::=	*binary _constant*
	|	*octal_constant*
	|	*decimal_constant*
	|	*hex_constant*
binary _constant	::=	*% binary_digits*
binary _digits	::=	*binary_digit*
	|	*binary_digit binary _digits*
binary_digit	::=	**0 – 1**
octal_constant	::=	*@ octal_digits*
octal_digits	::=	*octal_digit*
	|	*octal_digit octal_digits*
octal_digit	::=	**0-7**
decimal_constant	::=	*decimal_digits*
	|	*decimal_digit decimal_digits*
decimal_digits	::=	**0-9**
hex_constant	::=	**$** *hex_digits*
hex_digits	::=	*hex_digits*
	|	*hex_digit hex_digits*
hex_digit	::=	**0-9 | a-f | A-F**
string_constant	::=	*'string'*
	|	*"string"*

NOTE: A single quoted string cannot contain a single quote, and a double quoted string cannot contain a double quote.

string	::=	**printable_character**
	|	**printable_character** string
binary_operator	::=	+ | - | * | / | & | ^ | ! | % | << | >> | < | > | <= | >= | == | !=
unary_operator	::=	+ | - | ~
start_group_symbol	::=	(
end_group_sysbol	::=	)

memory_operator ::= page

NOTICE: These standards do not purport to address safety issues, if any associated with their use. It is the responsibility of the user of these standards to establish appropriate safety, health practices, and determine the applicability of regulatory limitations prior to use. Motorola makes no warranties or representations as to the suitability of the standards set forth herein for any particular application. The determination of the suitability of the standard is solely the responsibility of the user. Users are cautioned to refer to manufacturer's instructions, product labels, product data sheets, and other relevant literature respecting any materials mentioned herein. These standards are subject to change without notice.

The user's attention is called to the possibility that compliance with this standard may require use of copyrighted material or of an invention covered by patent rights. By publication of this standard, Motorola takes no position respecting the validity of any patent rights or copyrights asserted in connection with any item mentioned in this standard. Users of this standard are expressly advised that determination of any such patent rights or copyrights, and the risk of infringement of such rights, are entirely their own responsibility.

APPENDIX A. MCU HC05

A.1. INSTRUCTION MNEMONICS FOR THE CPU05

The instructions for the CPU05 use the following mnemonics with the associated operand forms. Instruction mnemonics are case insensitive. Legal syntax for each operand form is shown in a previous table.

CPU05 Mnemonic	*Operand Forms*
adc	imm8 | dir | ext | ind0 | ind1 | ind2
add	imm8 | dir | ext | ind0 | ind1 | ind2
and	imm8 | dir | ext | ind0 | ind1 | ind2
asl	dir | ind0 | ind1
asla	inh
aslx	inh
asr	dir | ind0 | ind1
asra	inh
asrx	inh
bcc	rel8
bclr	(dir), #mask8
bcs	rel8
beq	rel8
bge	rel8
bgt	rel8
bhcc	rel8
bhcs	rel8
bhi	rel8
bhs	rel8
bih	rel8
bil	rel8
bit	imm8 | dir | ext | ind0 | ind1 | ind2
blo	(alias of mnemonic bcs)
bls	rel8
bmc	rel8
bmi	rel8
bms	rel
bne	rel8
bpl	rel8
bra	rel8

CPU05 Mnemonic	*Operand Forms*
brclr	(dir), #mask8, rel8
brn	rel8
brset	(dir), #mask8, rel8
bset	(dir), #mask8
bsr	rel8
clc	inh
cli	inh
clr	dir | ind1
clra	inh
clrx	inh
clrh	inh
cmp	dir | inh | ext | ind0 | ind1 | ind2
com	dir | ind0 | ind1
coma	inh
comx	inh
cpx	Imm8 | dir | ext | ind0 | ind1 | ind2
dec	dir | ind0 | ind1
deca	inh
decx	inh
eor	imm8 | dir | ext | ind0 | ind1 | ind2
inc	dir | ind0 | ind1
inca	inh
incx	inh
jmp	dir | ext | ind0 | ind1 | ind2
jsr	dir | ext | ind0 | ind1 | ind2
lda	imm8 | dir | ext | ind0 | ind1 | ind2
ldx	Imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
lsl	(alias of mnemonic asl)
lsla	(alias of mnemonic asla)
lslx	(alias of mnemonic aslx)
lsr	dir | ind0 | ind1
lsra	inh
lsrx	inh
mul	inh
neg	dir | ind0 | ind1
nega	inh
negx	inh
nop	inh
ora	imm8 | dir | ext | ind0 | ind1 | ind2

CPU05 Mnemonic	*Operand Forms*
rol	dir | ind0 | ind1
rola	inh
rolx	inh
ror	dir | ind0 | ind1 | indi0
rora	inh
rorx	inh
rsp	inh
rti	inh
rts	inh
sbc	imm8 | dir | ext | ind0 | ind1 | ind2
sec	inh
sei	inh
sta	dir | ext | ind0 | ind1 | ind2
stop	inh
stx	dir | ext | ind0 | ind1 | ind2
sub	imm8 | dir | ext | ind0 | ind1 | ind2
swi	inh
tax	inh
tst	dir | ind0 | ind1
tsta	inh
tstx	inh
txa	inh
wait	inh

The **HC05 Reference Manual** gives the authoritative formats for CPU05 instructions.

A.2. CPU05 RESERVED IDENTIFIERS

Software developers should assume that assemblers for the 68HC05 family of microprocessors reserve the following identifiers: **ccr, a, x, sp,** and **pc.** All register mnemonics shall be case insensitive.

Motorola Microcontroller Division
6501 William Cannon Drive West, Austin, TX 78735
Phone: (512) 895-7634, FAX: (512) 895-1902

Document Number: 1001
Revision: 1.0
Date: 26 October, 1999

APPENDIX B. MCU HC08 FAMILY

B.1. INSTRUCTION MNEMONICS FOR THE CPU08

The instructions for the CPU08 use the following mnemonics with the associated operand forms. Instruction mnemonics are case insensitive. Legal syntax for each operand form is shown in a previous table.

CPU08 Mnemonic	*Operand Forms*
adc	imm8 | dir | ext | ind0 | ind1 | ind2 | indi2
add	imm8 | dir | ext | ind0 | ind1 | ind2 | indi2
ais	imm8
aix	imm8
and	imm8 | dir | ext | ind0 | ind1 | ind2 | indi2
asl	dir | ind0 | ind1
asla	inh
aslx	inh
asr	dir | ind0 | ind1
asra	inh
asrl	inh
bcc	rel8
bclr	(dir), #mask8
bcs	rel8
beq	rel8
bge	rel8
bgt	rel8
bhcc	rel8
bhcs	rel8
bhi	rel8
bhs	(alias of mnemonic bcc)
bih	rel8
bil	rel8
bit	imm8 | dir | ext | ind0 | ind1 | ind2 | indi2
ble	rel8
blo	(alias of mnemonic bcs)
bls	rel8
blt	rel8
bmi	rel8
bms	rel

CPU08 Mnemonic	*Operand Forms*
bne	rel8
bpl	rel8
bra	rel8
brclr	(dir), #mask8, rel8
brn	rel8
brset	(dir), #mask8, rel8
bset	(dir), #mask8
bsr	rel8
cbeq	dir | imm8 |ind0 | ind1
clc	(alias of instruction andcc #$fe)
cli	(alias of instruction andcc #$ef)
clr	dir | ind0 | ind1 | indi0 | indi2
clra	inh
clrx	inh
clrh	inh
cmp	dir | inh | ind0 | ind1 | ind2 | indi0 | indi2
com	dir | ind0 | ind1 | indi0
coma	inh
comx	inh
cphx	imm8 | dir
cpx	Imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
daa	inh
dbnz	dir | ind0 | ind1 | indi0
dbnza	inh
dbnzx	inh
dec	dir | ind0 | ind1 | indi0
deca	inh
decx	inh
div	inh
eor	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
inc	dir | ind0 | ind1 | indi0
inca	inh
incx	inh
jmp	dir | ext | ind0 | ind1 | ind2
jsr	dir | ext | ind0 | ind1 | ind2
lda	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ldhx	imm8 | dir
ldx	Imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
lsl	(alias of mnemonic asl)

CPU08 Mnemonic	*Operand Forms*
lsla	(alias of mnemonic asla)
lslx	(alias of mnemonic aslx)
lsr	dir | ind0 | ind1 | indi0
lsra	inh
lsrx	inh
mov	(imm8 | ext | ind0), (ext | ind0)
mul	inh
neg	dir | ind0 | ind1 | indi0
nega	inh
negx	inh
nop	inh
nsa	inh
ora	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
psha	inh
pshh	inh
pshx	inh
pula	inh
pulh	inh
pulx	inh
rol	dir | ind0 | ind1 | indi0
rola	inh
rolx	inh
ror	dir | ind0 | ind1 | indi0
rora	inh
rorx	inh
rsp	inh
rti	inh
rts	inh
sbc	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sec	(alias of instruction orcc #$01)
sei	(alias of instruction orcc #$10)
sta	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sthx	dir
stop	inh
stx	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sub	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
swi	inh
tap	inh
tax	inh

CPU08 Mnemonic	*Operand Forms*
tpa	inh
tst	dir | ind0 | ind1 | indi0
tsta	inh
tstx	inh
tsx	inh
txa	inh
txs	inh
wait	inh

The **HC08 CPU08 Reference Manual** gives the authoritative formats for CPU08 instructions.

B.2. CPU08 RESERVED IDENTIFIERS

Software developers should assume that assemblers for the 68HC08 family of microprocessors reserve the following identifiers: **ccr, a, h:x, sp,** and **pc.** All register mnemonics shall be case insensitive.

APPENDIX C. MCU HC11

C.1. INSTRUCTION MNEMONICS FOR THE CPU11

The instructions for the CPU11 use the following mnemonics with the associated operand forms. Instruction mnemonics are case insensitive. Legal syntax for each operand form is shown in a previous table.

CPU11 Mnemonic	*Operand Forms*
aba	inh
abx	inh
aby	inh
adca	imm8 | dir | ext | ind0 | ind1
adcb	imm8 | dir | ext | ind0 | ind1
adda	imm8 | dir | ext | ind0 | ind1
addb	imm8 | dir | ext | ind0 | ind1
addd	imm16 | dir | ext | ind0 | ind1
anda	imm8 | dir | ext | ind0 | ind1
andb	imm8 | dir | ext | ind0 | ind1
asl	ext | ind0 | ind1
asla	inh
aslb	inh
asld	inh
asr	ext | ind0 | ind1
asra	inh
asrb	inh
bcc	rel8
bclr	(dir | ext | ind0 | ind1), #mask8
bcs	rel8
beq	rel8
bge	rel8
bgt	rel8
bhi	rel8
bhs	rel8
bita	imm8 | dir | ext | ind0 | ind1
bitb	imm8 | dir | ext | ind0 | ind1
ble	rel8
blo	rel8
bls	rel8

CPU11 Mnemonic	*Operand Forms*
blt	rel8
bmi	rel8
bne	rel8
bpl	rel8
bra	rel8
brclr	(dir | ind0 | ind1), #mask8, rel8
brn	rel8
brset	(dir | ind0 | ind1), #mask8, rel8
bset	(dir | ind0 | ind1), #mask8
bsr	rel8
bvc	rel8
bvs	rel8
cba	inh
clc	inh
cli	inh
clr	ext | ind0 | ind1
clra	inh
clrb	inh
clv	inh
cmpa	imm8 | dir | ext | ind0 | ind1
cmpb	imm8 | dir | ext | ind0 | ind1
com	ext | ind0 | ind1
coma	inh
comb	inh
cpd	Imm16 | dir | ext | ind0 | ind1
cpx	Imm16 | dir | ext | ind0 | ind1
cpy	Imm16 | dir | ext | ind0 | ind1
daa	inh
dec	ext | ind0 | ind1
deca	inh
decb	inh
des	inh
dex	inh
dey	inh
eora	imm8 | dir | ext | ind0 | ind1
eorb	imm8 | dir | ext | ind0 | ind1
fdiv	inh
idiv	inh
inc	ext | ind0 | ind1

CPU11 Mnemonic	*Operand Forms*
inca	inh
incb	inh
ins	inh
inx	inh
iny	inh
jmp	ext | ind0 | ind1
jsr	dir | ext | ind0 | ind1
ldaa	imm8 | dir | ext | ind0 | ind1
ldab	imm8 | dir | ext | ind0 | ind1
ldd	imm16 | dir | ext | ind0 | ind1
lds	imm16 | dir | ext | ind0 | ind1
ldx	imm16 | dir | ext | ind0 | ind1
ldy	imm16 | dir | ext | ind0 | ind1
lsl	ext | ind0 | ind1
lsla	inh
lslb	inh
lsld	inh
lsr	ext | ind0 | ind1
lsra	inh
lsrb	inh
lsrd	inh
mul	inh
neg	ext | ind0 | ind1
nega	inh
negb	inh
nop	inh
oraa	imm8 | dir | ext | ind0 | ind1
orab	imm8 | dir | ext | ind0 | ind1
psha	inh
pshb	inh
pshx	inh
pshy	inh
pula	inh
pulb	inh
pulx	inh
puly	inh
rol	ext | ind0 | ind1
rola	inh
rolb	inh

CPU11 Mnemonic	*Operand Forms*
ror	ext | ind0 | ind1
rora	inh
rorb	inh
rti	inh
rts	inh
sba	inh
sbca	imm8 | dir | ext | ind0 | ind1
sbcb	imm8 | dir | ext | ind0 | ind1
sec	inh
sei	inh
sev	inh
staa	dir | ext | ind0 | ind1
stab	dir | ext | ind0 | ind1
std	dir | ext | ind0 | ind1
stop	inh
sts	dir | ext | ind0 | ind1
stx	dir | ext | ind0 | ind1
sty	dir | ext | ind0 | ind1
suba	imm8 | dir | ext | ind0 | ind1
subb	imm8 | dir | ext | ind0 | ind1
subd	imm16 | dir | ext | ind0 | ind1
swi	inh
tab	inh
tap	inh
tba	inh
test	inh
tpa	inh
tst	ext | ind0 | ind1
tsta	inh
tstb	inh
tsx	inh
tsy	inh
txs	inh
tys	inh
wai	inh
xgdx	inh
xgdy	inh

The **CPU11 Reference Manual** gives the authoritative formats for CPU11 instructions.

C.2. CPU11 RESERVED IDENTIFIERS

Software developers should assume that assemblers for the 68HC11 family of microprocessors reserve the following identifiers: **ccr, a, b, d, x, y, sp,** and **pc.** All register mnemonics shall be case insensitive.

Motorola Microcontroller Division
6501 William Cannon Drive West, Austin, TX 78735
Phone: (512) 895-7634, FAX: (512) 895-1902

Document Number: 1001
Revision: 1.0
Date: 26 October, 1999

APPENDIX D. MCU HC12

D.1. INSTRUCTION MNEMONICS FOR THE CPU12

The instructions for the CPU12 use the following mnemonics with the associated operand forms. Instruction mnemonics are case insensitive. Legal syntax for each operand form is shown in a previous table.

CPU12 Mnemonic	*Operand Forms*
aba	inh
abx	(alias of instruction leax b,x)
aby	(alias of instruction leay b,y)
adca	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
adcb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
adda	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
addb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
addd	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
anda	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
andb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
andcc	imm8
asl	ext | ind0 | ind1 | ind2 | indi0 | indi2
asla	inh
aslb	inh
asld	inh
asr	ext | ind0 | ind1 | ind2 | indi0 | indi2
asra	inh
asrb	inh
bcc	rel8
bclr	(dir | ext | ind0 | ind1 | ind2), #mask8
bcs	rel8
beq	rel8
bge	rel8
bgnd	inh
bgt	rel8
bhi	rel8
bhs	(alias of mnemonic bcc)
bita	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
bitb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ble	rel8

CPU12 Mnemonic	*Operand Forms*
blo	(alias of mnemonic bcs)
bls	rel8
blt	rel8
bmi	rel8
bne	rel8
bpl	rel8
bra	rel8
brclr	(dir | ext | ind0 | ind1 | ind2), #mask8, rel8
brn	rel8
brset	(dir | ext | ind0 | ind1 | ind2), #mask8, rel8
bset	(dir | ext | ind0 | ind1 | ind2), #mask8
bsr	rel8
bvc	rel8
bvs	rel8
call	(ext | ind0 | ind1 | ind2), page | indi0 | indi2
cba	inh
clc	(alias of instruction andcc #$fe)
cli	(alias of instruction andcc #$ef)
clr	ext | ind0 | ind1 | ind2 | indi0 | indi2
clra	inh
clrb	inh
clv	(alias of instruction andcc #$fd)
cmpa	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
cmpb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
com	ext | ind0 | ind1 | ind2 | indi0 | indi2
coma	inh
comb	inh
cpd	Imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
cps	Imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
cpx	Imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
cpy	Imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
daa	inh
dbeq	rel9
dbne	rel9
dec	ext | ind0 | ind1 | ind2 | indi0 | indi2
deca	inh
decb	inh
des	(alias of instruction leas -1,sp)
dex	inh

CPU12 Mnemonic	*Operand Forms*
dey	inh
ediv	inh
emacs	ext
edivs	inh
emaxd	ind0 | ind1 | ind2 | indi0 | indi2
emaxm	ind0 | ind1 | ind2 | indi0 | indi2
emind	ind0 | ind1 | ind2 | indi0 | indi2
eminm	ind0 | ind1 | ind2 | indi0 | indi2
emul	inh
emuls	inh
eora	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
eorb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
etbl	ind0
exg	txg
fdiv	inh
ibeq	rel9
ibne	rel9
idiv	inh
idivs	inh
inc	ext | ind0 | ind1 | ind2 | indi0 | indi2
inca	inh
incb	inh
ins	(alias of instruction leas 1,sp)
inx	inh
iny	inh
jmp	ext | ind0 | ind1 | ind2 | indi0 | indi2
jsr	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
lbcc	rel16
lbcs	rel16
lbeq	rel16
lbge	rel16
lbgt	rel16
lbhi	rel16
lbhs	(alias of mnemonic lbcc)
lble	rel16
lblo	(alias of mnemonic lbcs)
lbls	rel16
lblt	rel16
lbmi	rel16

CPU12 Mnemonic	*Operand Forms*
lbne	rel16
lbpl	rel16
lbra	rel16
lbrn	rel16
lbvc	rel16
lbvs	rel16
ldaa	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ldab	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ldd	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
lds	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ldx	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ldy	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
leas	ind0 | ind1 | ind2
leax	ind0 | ind1 | ind2
leay	ind0 | ind1 | ind2
lsl	(alias of mnemonic asl)
lsla	(alias of mnemonic asla)
lslb	(alias of mnemonic aslb)
lsld	(alias of mnemonic asld)
lsr	ext | ind0 | ind1 | ind2 | indi0 | indi2
lsra	inh
lsrb	inh
lsrd	inh
maxa	ind0 | ind1 | ind2 | indi0 | indi2
maxm	ind0 | ind1 | ind2 | indi0 | indi2
mem	inh
mina	ind0 | ind1 | ind2 | indi0 | indi2
minm	ind0 | ind1 | ind2 | indi0 | indi2
movb	(imm8 | ext | ind0), (ext | ind0)
movw	(imm16 | ext | ind0), (ext | ind0)
mul	inh
neg	ext | ind0 | ind1 | ind2 | indi0 | indi2
nega	inh
negb	inh
nop	inh
oraa	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
orab	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
orcc	imm8
psha	inh

CPU12 Mnemonic	*Operand Forms*
pshb	inh
pshc	inh
pshd	inh
pshx	inh
pshy	inh
pula	inh
pulb	inh
pulc	inh
puld	inh
pulx	inh
puly	inh
rev	inh
revw	inh
rol	ext | ind0 | ind1 | ind2 | indi0 | indi2
rola	inh
rolb	inh
ror	ext | ind0 | ind1 | ind2 | indi0 | indi2
rora	inh
rorb	inh
rto	inh
rti	inh
rts	inh
sba	inh
sbca	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sbcb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sec	(alias of instruction orcc #$01)
sei	(alias of instruction orcc #$10)
sev	(alias of instruction orcc #$02)
sex	sxg
staa	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
stab	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
std	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
stop	inh
sts	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
stx	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sty	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
suba	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
subb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
subd	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2

CPU12 Mnemonic	*Operand Forms*
swi	inh
tab	inh
tap	(alias of instruction tfr a,ccr)
tba	inh
tbeq	rel9
tbl	ind0
tbne	rel9
tfr	txg
tpa	(alias of instruction tfr ccr,a)
trap	inh
tst	ext | ind0 | ind1 | ind2 | indi0 | indi2
tsta	inh
tstb	inh
tsx	(alias of instruction tfr sp,x)
tsy	(alias of instruction tfr sp,y)
txs	(alias of instruction tfr x,sp)
tys	(alias of instruction tfr y,sp)
wai	inh
wav	inh
xgdx	(alias of instruction exg d,x)
xgdy	(alias of instruction exg d,y)

The **HC12 CPU12 Reference Manual** gives the authoritative formats for CPU12 instructions.

D.2. CPU12 RESERVED IDENTIFIERS

Software developers should assume that assemblers for the 68HC12 family of microprocessors reserve the following identifiers: **ccr, a, b, d, x, y, sp,** and **pc.** All register mnemonics shall be case insensitive.

Motorola Microcontroller Division
6501 William Cannon Drive West, Austin, TX 78735
Phone: (512) 895-7634, FAX: (512) 895-1902

Document Number: 1001
Revision: 1.0
Date: 26 October, 1999

APPENDIX E. MCU HC16

E.1. INSTRUCTION MNEMONICS FOR THE CPU16

The instructions for the CPU16 use the following mnemonics with the associated operand forms. Instruction mnemonics are case insensitive. Legal syntax for each operand form is shown in a previous table.

CPU16 Mnemonic	*Operand Forms*
aba	inh
abx	inh
aby	inh
abz	inh
ace	inh
aced	inh
adca	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
adcb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
adcd	Imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
adce	Imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
adda	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
addb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
addd	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
adde	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ade	inh
adx	inh
ady	inh
adz	inh
aex	inh
aey	inh
aez	inh
ais	imm8 | imm16
aix	imm8 | imm16
aiy	imm8 | imm16
aiz	imm8 | imm16
anda	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
andb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
andd	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ande	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
andp	imm16

CPU16 Mnemonic	*Operand Forms*
asl	ext | ind0 | ind1 | ind2 | indi0 | indi2
asla	inh
aslb	inh
asld	inh
asle	inh
aslm	inh
aslw	ext | ind0 | ind1 | ind2
asr	ext | ind0 | ind1 | ind2 | indi0 | indi2
asra	inh
asrb	inh
asrd	inh
asre	inh
asrm	inh
asrw	ext | ind0 | ind1 | ind2
bcc	rel8
bclr	(dir | ext | ind0 | ind1 | ind2), #mask8
bcs	rel8
beq	rel8
bge	rel8
bgnd	inh
bgt	rel8
bhi	rel8
bita	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
bitb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ble	rel8
bls	rel8
blt	rel8
bmi	rel8
bne	rel8
bpl	rel8
bra	rel8
brclr	(dir | ext | ind0 | ind1 | ind2), #mask8, rel8
brn	rel8
brset	(dir | ext | ind0 | ind1 | ind2), #mask8, rel8, rel16
bset	(dir | ext | ind0 | ind1 | ind2), #mask8
bsetw	(dir | ext | ind0 | ind1 | ind2), #mask16
bsr	rel8
bvc	rel8
bvs	rel8

CPU16 Mnemonic	*Operand Forms*
cba	inh
clr	ext | ind0 | ind1 | ind2 | indi0 | indi2
clra	inh
clrb	inh
clrd	inh
clre	inh
clrm	inh
clrw	ext | ind0 | ind1 | ind2
cmpa	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
cmpb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
com	ext | ind0 | ind1 | ind2 | indi0 | indi2
coma	inh
comb	inh
comd	inh
come	inh
comw	ext | ind0 | ind1 | ind2
cpd	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
cpe	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
cps	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
cpx	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
cpy	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
cpz	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
daa	inh
dec	ext | ind0 | ind1 | ind2 | indi0 | indi2
deca	inh
decb	inh
decw	ext | ind0 | ind1 | ind2
ediv	inh
edivs	inh
emul	inh
emuls	inh
eora	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
eorb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
eord	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
eore	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
fdiv	inh
fmuls	inh
idiv	inh
inc	ext | ind0 | ind1 | ind2 | indi0 | indi2
inca	inh

CPU16 Mnemonic	*Operand Forms*
incb	inh
incw	ext | ind0 | ind1 | ind2
jmp	ext | ind0 | ind1 | ind2 | indi0 | indi2
jsr	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
lbcc	rel16
lbcs	rel16
lbeq	rel16
lbev	rel16
lbge	rel16
lbgt	rel16
lbhi	rel16
lble	rel16
lbls	rel16
lblt	rel16
lbmi	rel16
lbmv	rel16
lbne	rel16
lbpl	rel16
lbra	rel16
lbm	rel16
lbsr	rel16
lbvc	rel16
lbvs	rel16
ldaa	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ldab	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ldd	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
lde	imm16 | ext | ind0 | ind1 | ind2 | indi0 | indi2
lded	ext
ldhi	ext
lds	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ldx	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ldy	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ldz	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
lpstop	inh
lsl	ext | ind0 | ind1 | ind2 | indi0 | indi2
lsla	inh
lslb	inh
lsld	inh
lsle	inh

CPU16 Mnemonic	*Operand Forms*
lslm	inh
lslw	ext | ind0 | ind1 | ind2
lsr	ext | ind0 | ind1 | ind2 | indi0 | indi2
lsra	inh
lsrb	inh
lsrd	inh
lsre	inh
lsrw	ext | ind0 | ind1 | ind2
mac	imm8
movb	(imm8 | ext | ind0), (ext | ind0)
movw	(imm16 | ext | ind0), (ext | ind0)
mul	inh
neg	ext | ind0 | ind1 | ind2 | indi0 | indi2
nega	inh
negb	inh
negd	inh
nege	inh
negw	ext | ind0 | ind1 | ind2
nop	inh
oraa	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
orab	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ord	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ore	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
orp	imm16
psha	inh
pshb	inh
pshm	imm8
pshmac	inh
pula	inh
pulb	inh
pulm	imm8
pulmac	inh
rol	ext | ind0 | ind1 | ind2 | indi0 | indi2
rola	inh
rolb	inh
rold	inh
role	inh
rolw	ext | ind0 | ind1 | ind2
ror	ext | ind0 | ind1 | ind2 | indi0 | indi2

CPU16 Mnemonic	*Operand Forms*
rora	inh
rorb	inh
rord	inh
rore	inh
rorw	ext | ind0 | ind1 | ind2
rti	inh
rts	inh
sba	inh
sbca	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sbcb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sbcd	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sbce	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sde	inh
staa	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
stab	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
std	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
ste	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sted	ext
sts	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
stx	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sty	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
stz	dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
suba	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
subb	imm8 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
subd	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
sube	imm16 | dir | ext | ind0 | ind1 | ind2 | indi0 | indi2
swi	inh
sxt	inh
tab	inh
tap	inh
tba	inh
tbek	inh
tbsk	inh
tbxk	inh
tbyk	inh
tbzk	inh
tde	inh
tdmsk	inh
tdp	inh

CPU16 Mnemonic	*Operand Forms*
ted	inh
tedm	inh
tekb	inh
tem	inh
tmer	inh
tmet	inh
tmxed	inh
tpa	inh
tpd	inh
tskb	inh
tst	ext | ind0 | ind1 | ind2 | indi0 | indi2
tsta	inh
tstb	inh
tstd	inh
tste	inh
tstw	ext | ind0 | ind1 | ind2
tsx	inh
tsy	inh
tsz	inh
txkb	inh
txs	inh
txy	inh
txz	inh
tykb	inh
tys	inh
tyx	inh
tyz	inh
tzkb	inh
tzs	inh
tzx	inh
tzy	inh
wai	inh
xgab	inh
xgde	inh
xgdx	inh
xgdy	inh
xgdz	inh
xgex	inh
xgey	inh

CPU16 Mnemonic	*Operand Forms*
xgez	inh

The **CPU16 Reference Manual** gives the authoritative formats for CPU16 instructions.

E.2. CPU16 RESERVED IDENTIFIERS

Software developers should assume that assemblers for the 68HC16 family of microprocessors reserve the following identifiers: **ccr, a, b, d, e, x, y, z, sp, pc, ek, xk, yk, zk, pk, sk, hr, ir,** and **am.** All register mnemonics shall be case insensitive.