ADEV11 Development System for AmigaDOS
Users's Manual

Version 1.00

A 68HC11 cross compiler (C), assembler, linker, librarian and downloader for the Amiga
Public Domain

Stan Burton
1978 26 St. SE
Medicine Hat, Alta, CANADA
T1A 2G8

ADEV11 Development System for AmigaDOS
Command Reference

Version 1.0

Table of Contents

NAME

HCLoad

SYNOPSIS

```
HCLoad [options] file_name
```

DESCRIPTION

HCLoad is a utility for downloading an S-record file to the 68HC11 EEPROM. It first downloads a file called HCBOOT to the ram of the device. This program is an S-record loader that executes when loaded and receives the S-records from the specified file and programs them into the EEPROM.

The HCBOOT file must be found in your current directory. This file is different for some versions of the processor. Three versions are provided; one for the 68HC811, 68HC11F1 and one for the 68HC11F1 which enables programming of the CONFIG register.

Currently, HCLoad only has one option:

-f<number>

for use where the E clock frequency of the HC11 is not 2.0000 MHz. <number> is the E clock frequency of the HC11 in Hz.

EXAMPLE

The following line downloads the file download.sr to an HC11 with a crystal frequency of 4.9152 MHz (E clock = 1.2288 MHz).

```
HCload -f1228800 download.sr
```

NAME

SAsm

SYNOPSIS

```
SAsm [options] <srcfile> [options]
```

DESCRIPTION

SAsm is the assembler for the DEV11 system. It is a high level macro cross assembler for the Motorola 6803, 6805, 68HC11 and 68HC16 families and for the Hitachi 6303 family. It is a highly modified version of the publicly distributable DASM V2.12. The 68HC16 code generation has not been well tested since I do not have an HC16 to test it on. If you find any problems with it let me know (Stan).

SAsm produces a relocatable file which can be linked together (Slink) with other modules and/or library elements to produce an executable file. Naturally this includes the ability to have multiple segments within a module and BSS or uninitialized segments.

COMMAND LINE

srcfile: if no extension is specified in the name, .a is added
outfile: the file generated is the extension-less srcfile name with a .o extension

The following options are available:

-l[name]	generate list file, if no name is specified extension-less srcfile with .lst extension is used	
-s[name]	generate symbol file, if no name is specified extension-less srcfile with .sym extension is used	
-v#	select verboseness 0-4 (default 0, see below)	
	0	(default) Only warnings and errors are generated
	1	Segment list information is generated after each pass, Include file names are displayed and reasons why another pass is required are given.
	2	Mismatches between program labels and equates are displayed on every pass (usually none occur in the first pass unless you have re-declared a symbol name).
	3	Unresolved and unreferenced symbols are displayed every pass

	(unsorted, sorry)
4	An entire symbol list is displayed every pass to STDOUT. (unsorted, sorry)

-p#	select number of passes 2-9 (default 2)
-d	debug mode
-DSYMBOL	predefine a symbol, set to 0
-DSYMBOL=EXPRESSION	predefine a symbol, set to expression

Example: asm master.asm -lram:list -v3 -DVER=4

The verbose options can provide additional information about reasons for failure to assemble a file. Part of this is the reason code:

R1,R2 reason code: R3

where	R1	is the number of times the assembler encountered something requiring another pass to resolve.
	R2	is the number of references to unknown symbols which occured in the pass (but only R1 determines the need for another pass).
	R3	is a BITMASK of the reasons why another pass is required. See the end of this document for bit designations.

-expressions, as in C. (all expressions are computed with 32 bit integers)
-no real limitation on label size, label values are 32 bits.
-complex pseudo ops, repeat loops, macros, etc....

The following special characters are used in the symbol dump:

????	unknown value
str	symbol is a string
eqm	symbol is an eqm macro
(r)	symbol has been referenced
(s)	symbol created with SET or EQM pseudo-op

LABELS and SYMBOLS

A label consists of one or more characters from the set A-Z, a-z, 0-9. The first character must be alphabetic. There is no limit to the name length. The value the label assumes the range of a 32 bit integer.

The label will be set to the current segment counter either before or after a pseudo-op is executed. Most of the time, the label is set before the pseudo-op is executed. The following pseudo-op's labels are created AFTER execution of the pseudo-op:

SEG, ALIGN

PROCESSOR MODEL

The processor model is chosen with the PROCESSOR pseudo-op and should be the first thing you do in your assembly file. Only one PROCESSOR pseudo-op may be declared in the entire assembly.

SEGMENTS

The SEG pseudo-op creates/sets the current segment. Segments are used to separate the various parts of a program, for example CODE and BSS_DATA; later the linker could place the CODE in the EPROM address range and the data at the ram address range. The use of DS or RMB statements in a segment planned for ROM is not logical.

As a result of the use of relocatable segments the ORG statement found in simpler assemblers is not used; its functionality is passed to the linker.

’Uninitialized’ (.U) segments do not produce output. Therefore, output generating statements are not allowed in these segments.

MACROS

You cannot have a macro definition within a macro definition, but can nest macro calls.

Arguments passed to macros are referenced with: {#}. The first argument passed to a macro would thus be {1}. You should always use LOCAL labels (.name) inside macros which you use more than once. {0} represents an EXACT substitution of the ENTIRE argument line.

GENERAL

? The other major feature in this assembler is the SUBROUTINE pseudo-op, which logically separates local labels (starting with a dot). This allows you to reuse label names (for example, .1 .fail) rather than think up crazy combinations of the current subroutine to keep it all unique.

PSEUDOPS

```
        INCLUDE  "name"
                        Include another assembly file.

[label] SEG[.U]     name
[label] RSEG[.U]     name
                        This sets the current segment, creating it if neccessary.  If a .U extension is
                            specified on segment creation, the segment is an UNINITIALIZED
```

segment. The .U is not needed when going back to an already created uninitialized segment, though it makes the code more readable.

[label] DC[.BWL] exp,exp,exp ...
[label] FDB exp,exp,exp ...
[label] FCB exp,exp,exp ...

Declare data in the current segment. The default size extension for DC is a byte. FCB allows only byte size and FDB allows only word size.

[label] DS[.BWL] exp[,filler]
[label] RMB exp[,filler]

Declare space (default filler is 0). Note that the number of bytes generated is exp * entrysize (1,2, or 4). The default size extension for DS is a byte. RMB allows only the size of byte. Note that the default filler is always 0.

[label] DV[.BWL] eqmlabel exp,exp,exp....

This is equivalent to DC, but each exp in the list is passed through the symbolic expression specified by the EQM label. The expression is held in a special symbol dotdot '..' on each call to the EQM label.

See EQM below

[label] HEX hh hh hh..

This sets down raw HEX data. Spaces are optional between bytes. NO EXPRESSIONS are allowed. Note that you do NOT place a $ in front of the digits. This is a short form for creating tables compactly. Data is always layed down on a byte-by-byte basis.

Example: HEX 1A45 45 13254F 3E12

ERR

Abort assembly.

[label] XDEF symbol,symbol,symbol
[label] PUBLIC symbol,symbol,symbol

Defines which symbols/labels are available outside of this module.

[label] XREF symbol,symbol,symbol
[label] EXTERN symbol,symbol,symbol

Declares which symbols/labels from outside modules are used in this module.

PROCESSOR model

Do not quote. Model is one of: 6803,HD6303,68705,68HC11, 68HC16. Can only be executed once, and should be the first thing encountered by the assembler.

ECHO exp,exp,exp

The expressions (which may also be strings), are echod on the screen and into the list file

[label] ALIGN   N[,fill]

Align the current PC to an N byte boundry. The default fill character is always 0.

[label] SUBROUTINE name

This isn't really a subroutine, but a boundry between sets of temporary labels (which begin with a dot). Temporary label names are unique within segments of code bounded by SUBROUTINE:

```
CHARLIE subroutine
        ldx     #10
.1      dex
        bne     .1
BEN     subroutine
        ldx     #20
.qq     dex
        bne .qq
```

Automatic temporary label boundries occur for each macro level. Usually temporary labels are used in macros and within actual subroutines (so you don't have to think up a thousand different names)

symbolEQU   exp

The expression is evaluated and the result assigned to the symbol.

symbolEQM   exp

The STRING representing the expression is assigned to the symbol. Occurances of the symbol in later expressions causes the string to be evaluated for each occurance. Also used in conjuction with the DV psuedo-op.

symbolSET   exp

Same as EQU, but the symbol may be reassigned later.

END   [symbol]

Optional. If used with a symbol name the value of that symbol will be entered into the output file as the starting address of the program. May only be used once and only at the end of the file. Be careful - if a symbol is not given and a comment without a preceding ';' is used the first word of the comment will be interpreted as the symbol.

MAC name
MACRO name

Declare a macro. lines between MAC and ENDM are the macro. You cannot recursively declare a macro. You CAN recursively use a macro (reference a macro in a macro). No label is allowed to the left of MAC or ENDM.

ENDM

End of macro def. NO LABEL ALLOWED ON THE LEFT!

MEXIT

Used in conjuction with conditionals. Exits the current macro level.

[label] IFCONST exp
[label] IFD exp

Is TRUE if the expression result is defined, FALSE otherwise and NO error is generated if the expression is undefined.

[label] IFNCONST exp
[label] IFND exp

Is TRUE if the expression result is undefined, FALSE otherwise and NO error is generated if the expression is undefined.

[label] IF exp

Is TRUE if the expression result is defined AND non-zero. Is FALSE if the expression result is defined AND zero. Neither IF or ELSE will be executed if the expression result is undefined. If the expression is undefined, another assembly pass maybe required.

[label] ELSE

ELSE the current IF.

[label] ENDIF
[label] ENDC
[label] EIF

Terminate an IF. ENDIF and EIF are equivalent.

[label] REPEAT exp
[label] REPEND

Repeat code between REPEAT/REPEND 'exp' times. if exp == 0, the code repeats forever. exp is evaluated once.

```
Y       SET     0
        REPEAT  10
X       SET     0
        REPEAT  10
        DC      X,Y
X       SET     X + 1
        REPEND
Y       SET     Y + 1
        REPEND
```

generates an output table: 0,0 1,0 2,0 ... 9,0 0,1 1,1 2,1... 9,1, etc...

Labels within a REPEAT/REPEND should be temporary labels with a SUBROUTINE pseudoop to keep them unique.

The Label to the left of REPEND is assigned AFTER the loop FINISHES.

FORCED ADDRESSING MODES

[label] XXX[.force] operand

XXX is some mnemonic, not necessarily three characters long. The .FORCE optional extension is used to force specific addressing modes. Force extensions are also used with DS,DC, and DV to determine the element size.

example: lda.z charlie

Force	Description	Alternate Force Extension	
i	implied		
ind	indirect word		
0	implied		
b	byte address	z d	(zeropage, direct)
bx	byte address indexed x		
by	byte address indexed y		
w	word address	e a	(extended, absolute)
l	longword (4 bytes) (DS/DC/DV)		
r	relative		
u	uninitialized (SEG)		

EXPRESSIONS

Some operators, such as ||, can return a resolved value even if one of the expressions is not resolved. Operators are as follows:

NOTE WELL! Some operations will result in non-byte values when a byte value was wanted.
Example: ~1 is NOT $FF, but $FFFFFFFF.
Preceding it with a < (take LSB of) will solve the problem.
ALL ARITHMETIC IS CARRIED OUT IN 32 BITS. The final result will be automatically truncated to the maximum handleable by the particular machine language (usually a word) when applied to standard mnemonics.

PRECEDENCE		
	UNARY	
20	~exp	one's complement.
20	-exp	negation
20	!exp	not expression (returns 0 if exp non-zero, 1 if exp zero)
20	<exp	take LSB byte of a 16 bit expression
20	>exp	take MSB byte of an expression
	BINARY	
19	*	multiplication
19	/	division
19	%	mod
18	+	addition
18	-	subtraction
17	>>,<<	shift right, shift left
16	>,>=	greater, greater equal
16	<,<=	smaller, smaller equal
15	==	equal to. Try to use this instead of =
15	=	exactly the same as == (exists compatibility)
15	!=	not equal to
14	&	logical and
13	^	logical xor
12	|	logical or
11	&&	left expression is true AND right expression is true
10	||	left expression is true OR right expression is true
9	?	if left expression is true, result is right expression, else result is 0. [10 ? 20] returns 20
8	[]	group expressions
7	,	separate expressions in list (also used in addressing mode resolution, BE CAREFUL!

CONSTANTS

nnn	decimal
0nnn	octal
%nnn	binary
$nnn	hex
’c	character
’c’	character
"cc.."	string (NOT zero terminated if in DC/DS/DV)
[exp]d	the constant expressions is evaluated and it’s decimal result turned into an ascii string.

SYMBOLS

..	holds evaluated value in DV pseudo op
.name	represents a temporary symbol name. Temporary symbols may be reused inside MACROS and between SUBROUTINES, but may not be referenced across macros or across SUBROUTINEs.
.	current program counter (as of the beginning of the instruction).
name	beginning with an alpha character and containing letters, numbers, or ’_’. Represents some global symbol name.

WHY codes:

Each bit in the WHY word (verbose option 1) is a reason (why the assembler needs to do another pass), as follows:

Bit	Meaning
0	expression in mnemonic not resolved
1	
2	expression in a DC not resolved
3	expression in a DV not resolved (probably in DV’s EQM symbol)
4	expression in a DV not resolved (could be in DV’s EQM symbol)
5	expression in a DS not resolved
6	expression in an ALIGN not resolved
7	
8	???ALIGN: Normal origin not known (if in ORG at the time)
9	EQU: expression not resolved
10	EQU: value mismatch from previous pass (phase error)
11	IF: expression not resolved
12	REPEAT: expression not resolved
13	a program label has been defined after it has been referenced (forward reference) and thus we need another pass
14	a program label’s value is different from that of the previous pass (phase error)

NAME

SDis

SYNOPSIS

```
SDis <file name>
```

DESCRIPTION

SDis is the dis-assembler for the DEV11 system. It accepts a Motorola S-record format file and outputs a symbolic dis-assembled file to stdout, by default the console. The output file is assembler (SAsm) ready and would assemble to be exactly the same as the file that was dis-assembled.

SDis is a tracking dis-assembler which means that it must have the address of an executable instruction to start from. Normally it gets this from the S9 record if your file has one. Otherwise or if other addresses must be known, as might be the case with a monitor ROM with many entry points, the dis-assembler asks for an address(es) to start from. A tracking dis-assembler will not attempt to dis-assemble data; it knows the difference (but it can be intentionally fooled).

If SDis finds a place in the file that it believes is data but does not have a label, i.e. that address is not accessed by any of the code that it knows about, it will print out "??? no label."

NAME

slib

SYNOPSIS

```
slib [-a<1>] [-r<1>] [-l] [-c<1>] <library name>

    <1>  - one file
```

DESCRIPTION

Slib is the librarian for the DEV11 system. The following options are recognized by slib:

-a Specifies an object file that will be added to the library. The -a option may be used more than once.

-c Specifies a file that contains option commands to be processed by slib. The file may be broken into lines, but each line must start with an option. A command file can specify another command file and more than one command file can be used. An example file might look as shown below.

```
-adownload.o
-rcowtown.o
-l
```

-l Causes a listing of the library to be sent to the console (unless redirected). The listing includes the name of the module, its size (not the code size), and the symbols that are available from that module. The -l option can be used more than once.

-r Specifies an object module to be removed from the library. The -r option may be used more than once.

One and only one library must be specified for each invocation of slib.

NAME

slink

SYNOPSIS

```
slink [FROM <1+>] [TO <1>] [WITH <1>] [LIB <1+>] [MEM <range> <1+>]

    <1>  - one file
    <1+> - one or more files (separated by commas)
    <range> = <hex address>-<hex address>
```

DESCRIPTION

Slink is the linker for the DEV11 system. Slink accepts keyword commands to control the process of linking. Keywords are not case sensitive. The following keywords are recognized by slink:

FROM	Provides a list of object files that will become the root of the output file. FROM must be used once and may be used more than once with each use adding to the root, but you must specify at least one file for each use.
LIB	Provides a list of library files to be scanned to resolve symbols not found in the root files. Only the modules containing the unresolved symbols will be included in the output file. LIB can be used more than once.
MEM	Specifies an area of memory and provides a list of the segment names that are to be located in that area. The linker must know where to place every segment that is used. MEM is usually used more than once.
TO	Specifies the output file to create. The file will be an S-record format file. TO must be used once and only once.
WITH	Specifies a file that contains keyword commands to be processed by slink. The file may be broken into lines, but each line must start with a keyword. A WITH file can WITH another file and more than one WITH file may be used. An example file might look as shown below.

```
FROM download.o
TO download.sr
MEM 100-7fff EXT_RAM
MEM 8000-83ff INT_RAM
MEM 0-ff ZPAGE
MEM fe00-ffd5 EEPROM
MEM ffd6-ffff VECTORS
```

EXAMPLE

The following line invokes the linker using the command file download.sln and creating the linker map file download.map.

```
Slink MAP download.map WITH download.sln
```

ADEV11 Development System for AmigaDOS
Library Reference

Version 1.0

Function List

FUNCTION	**DESCRIPTION**	**LIBRARY**
asc2byte	**convert ASCII string to byte**	**HC11.lib**
find_spc	**find first space ' ' character in string**	**HC11.lib**
put_asc	**print out word size number (signed)**	**HC11.lib**
put_asc_sm	**print out byte size number (signed)**	**HC11.lib**
put_asc_u	**print out word size number (unsigned)**	**HC11.lib**
search	**search for string in string list**	**HC11.lib**
skip_spc	**find first non-space ' ' character in string**	**HC11.lib**
str_lookup	**find string in string list from ordinate**	**HC11.lib**

NAME

asc2byte

SYNOPSIS

IN:	X	pointer to string
OUT:	A	value result
	X	pointer to first non-numeric character
	B	modified
	Y	not modified

DESCRIPTION

This function converts an ASCII number in a string to a binary value. The conversion stops at the first non-numeric character.

NAME

find_spc
skip_spc

SYNOPSIS

(find_spc)

IN:	X	pointer to string
OUT:	X	pointer to space character or NULL
	A	space character or NULL
	B,Y	not modified

(skip_spc)

IN:	X	pointer to string
OUT:	X	pointer to non-space character
	A	non-space character
	B,Y	not modified

DESCRIPTION

These functions search through a string to find the presence or absence of a space.

NAME

put_asc
put_asc_sm
put_asc_u

SYNOPSIS

(put_asc, put_asc_u)

IN:	D	value to convert
OUT:	D,X	modified
	Y	not modified

(put_asc_sm)

IN:	A	value to convert
OUT:	D,X	modified
	Y	not modified

DESCRIPTION

These functions convert a binary number to ASCII and print it.

NAME

search
str_lookup

SYNOPSIS

(search)

```
IN:   X    pointer to search string
      Y    pointer to string list
OUT:  X    not modified
      Y    modified
      A    ordinate of string in list or -1
      B    not modified
```

(str_lookup)

```
IN:   X    pointer to string list
OUT:  X    pointer to string
      Y    not modified
      D    modified
```

DESCRIPTION

These function perform inverse actions on string lists. ‘search’ searches for a string in the list and returns its ordinal number (first string in list => 0). ‘str_lookup’ finds a string given the ordinal number.

A string list is a linear sequence of strings terminated by a null string, for example:

```
DC    "string0",0
DC    "string1",0
DC    "string2",0
DC    0
```