Cfasm ColdFire Assembler

rev. 1.09

PRELIMINARY

1. GENERAL

The ColdFire Assembler, cfasm, is a freeware assembler for ColdFire processors. Instruction set supported: 52xx, 5307 and 5407.

Command line arguments specify the filenames to assemble. Only one object file is output even though several files can be specified and assembled together. The assembler can output in Motorola SREC, raw binary or GeckoOS object code formats.

The object file is placed in the file '<inputfilename>.o' where 'inputfilename' was the name of the actual assembler file (missing any extensions). The listing and error messages are written to the standard output.

The listing file contains the address and bytes assembled for each line of input followed by the original input line (unchanged, but moved over to the right some). If an input line causes more than 6 bytes to be output (e.g. a long dc.b directive), additional bytes are listed on succeeding lines with no address preceding them.

Equates cause the value of the expression to replace the address field in the listing. Equates that have forward references cause phasing Errors in Pass 2.

It is unwise to have more than one assembly in progress per directory since the object file would be the same for all assemblies running.

PRELIMINARY

1.1 EXPRESSIONS

Expressions may consist of symbols, constants or the character '*' (denoting the current value of the program counter) joined together by one of the operators: +-*/%&|^<<>>~.

+	add
-	subtract
*	multiply
/	divide
%	modulo (remainder after division)
&	bitwise and
|	bitwise or
^	bitwise exclusive-or
<<	bitwise left shift
>>	bitwise right shift
~	ones complement

Expressions are evaluated left to right and there is no provision for parenthesized expressions. Arithmetic is carried out in signed 32-bit twos-complement integer precision.

1.2 CONSTANTS

Constants are constructed as follows:

0x	followed by hexadecimal constant
$	followed by hexadecimal constant
@	followed by octal constant
%	followed by binary constant
digit	decimal constant

String constants are specified by enclosing the string in single quotes. Strings are only recognized by the DC pseudo-op.

1.3 ERRORS

Error diagnostics are placed in the listing file just before the line containing the error. Format of the error line is:

```
Line_number: Description of error
        or
Line_number: Warning --- Description of error
```

Errors of the first type in pass one cause cancellation of pass two. Warnings do not cause cancellation of pass two but should cause you to wonder where they came from.

Error messages are meant to be self-explanatory. If more than one file is being assembled, the file name precedes the error:

```
File_name,Line_number: Description of error
```

Finally, some errors are classed as fatal and cause an immediate termination of the assembly. Generally these errors occur when a temporary file cannot be created or is lost during the assembly.

The exit status of the assembler ($status for csh users) is equal to the number of errors detected during the assembly.

1.4 PSEUDO OPS

The pseudo-ops (pseudo operations) are:

org	Origin (checks for word alignment)
equ	Equates
dc	Define constant(s)
dr	Define relative(s) (usually used for library offset tables)
ds	Define storage
opt	Options
section	set to either code, data or bss
xref *<symbol>*	external symbol reference. The symbol exists in another file.
xdef *<symbol>*	external symbol definition. This symbol can be referenced from another file.
idnt *<name>,<version>,<date>*	Identification strings. Each string will cause the generation of individual GeckoOS blocks of type BLOCK_NAME, BLOCK_VERSION and

PRELIMINARY

BLOCK_DATE (only if object code option enabled).

include <*file*> Filename of include file to be processed.

incdir Include directory. Include files are first searched in the current directory and then in the incdir directory.

incbin <*file*> Include a binary file into the assembly. This file is not assembled, but added as if it were raw data. Useful for tables, graphic images etc...

lvorst Reset library vector offset counter.

lvo <*symbol*> Library vector offset entry.

call <*symbol*> Call a library function using lvo, register A6 points to library base

push {reglist} Performs;

```
lea.l -x(a7),a7
movem.l {reglist},(a7)
```

where x is the number of registers*4

pull *{reglist}* Performs;

```
movem.l (a7),{reglist}
lea.l x(a7),a7
```

where x is the number of registers*4

structure <*symbol*>,<*value*> Creates a new symbol set to value. An internal variable is also set to the value.

stkstruct <*symbol*>,<*value*> Creates a new symbol set to value. An internal variable is also set to the value. The stack structure defines all further components to negative values.

struct <*symbol*>,<*value*> Creates a new symbol set to current internal value and the value is then added to the internal.

byte <*symbol*> Creates a new symbol set to current internal value and increments the internal value by one except if defined as a stack structure in which case the internal value is decremented first by one and then set to the symbol.

word <*symbol*> Creates a new symbol set to current internal value and increments the internal value by two except if defined as a stack structure in which case the internal value is decremented first by two and then set to the symbol.

long <*symbol*> Creates a new symbol set to current internal value and increments the internal value by four except if defined as a stack structure in which case the internal value is decremented first by four and then set to the symbol.

label <*symbol*> The symbol is set to the current internal value.

aptr <*symbol*> Absolute pointer, same as long.

rptr <*symbol*> Relative pointer, same as long.

ulong <*symbol*> Unsigned long, same as long.

fixed <*symbol*> Fixed point value, same as long.

float <*symbol*> Floating point value, same as long.

short <*symbol*> Short integer, same as word.

ushort <*symbol*> Unsigned short, same as word.

uword <*symbol*> Unsigned word, same as word.

char <*symbol*> Character, same as byte.

ubyte <*symbol*> Unsigned byte, same as byte.

bitdef <*prefix*>,<*symbol*>,<*bitnum*>

Creates two symbols based on the prefix. The first symbol is a bit version and the second is a mask version. e.g.

```
bitdef PA,STATUS,2
```

produces PAB_STATUS with the value 2 and PAF_STATUS with the value 1<<2 or 4.

```
bitdef PRE,DATA,7
```

produces PREB_DATA with the value 7 and PREF_DATA with the value

PRELIMINARY

1<<7 or 128.

1.4.1 CONDITIONAL

endc	Terminate a conditional assembly block.
else	Assemble code based on previous condition statement.
ifd <*symbol*>	If symbol defined, then assemble the next block of code until an 'endc' pseudo-op.
ifnd <*symbol*>	If symbol not defined, then assemble the next block of code until an 'endc' pseudo-op.
ifeq <*expr*>	If expression evalutes to zero, then assemble next code block.
ifne <*expr*>	If expression evalutes to non-zero, then assemble next code block.
ifge <*expr*>	If expression evalutes to >=0, then assemble next code block.
ifgt <*expr*>	If expression evalutes to >0, then assemble next code block.
ifle <*expr*>	If expression evalutes to <=0, then assemble next code block.
iflt <*expr*>	If expression evalutes to <0, then assemble next code block.

1.4.2 OPT pseudo-op

list	Turn on output listing
nolist	Turn off output listing (default)
brl	Force forward refs long (default is long)
object	Produce object code output (GeckoOS format)
binary	Produce binary output (overridden by object flag, raw format)
rtxnop	Adds NOP instruction after RTS or RTE instructions

The above pseudo-ops are treated as mnemonics, and must be preceded by at least one white space character in the assembly file.

Some of the more common pseudo-ops are not present:

spc	Use blank lines instead
pag[e]	Use a comment with cntrl-L imbeded
end	The assembly ends when there is no more input
ttl	use 'pr' to get headings and page numbers
nam[e]	

The above pseudo-ops are recognized, but ignored.

PRELIMINARY

2. DETAILS

Symbol: A string of characters with a non-initial digit. The string of characters may be from the set:

[a-z][A-Z]_[0-9]$

(_ count as a non-digit). The '$' counts as a digit to avoid confusion with hexadecimal constants. All characters of a symbol are significant, with upper and lower case characters being distinct. The maximum number of characters in a symbol is currently set at 30.

The symbol table can grow until the assembler runs out of memory.

Label: A symbol starting in the first column is a label and may optionally be ended with a ':'. A label may appear on a line by itself and is then interpreted as:

```
Label   EQU    *
```

Local: A local label is defined as any label ending in a '$'. This label is special in that it belongs only between two normal labels. e.g.

```
Start:
        nop
10$     bra     10$
Mid:
        nop
10$     bra     10$
        EndCode:
```

Mnemonic: A symbol preceded by at least one white space character. Upper case characters in this field are converted to lower case before being checked as a legal mnemonic. Thus 'nop', 'NOP' and even 'NoP' are recognized as the same mnemonic.

Operand: Follows mnemonic, separated by at least one white space character. The contents of the operand field is interpreted by each instruction.

White space: A blank or a tab

Comment: Any text after all operands for a given mnemonic have been processed or, a line beginning with '*' or '#' up to the end of line or, an empty line.

Continuations: If a line ends with a backslash (\) then the next line is fetched and added to the end of the first line. This continues until a line is seen which doesn't end in \ or until MAXBUF characters have been collected (MAXBUF >= 256).

2.1 BRANCHES

Branches can contain byte,word or long (V4 only) offsets. Word offset is the default for unspecified branches. E.g.;

```
bra     10$     ; Defaults to word offset
bra.b   10$     ; Byte offset
bra.w   10$     ; Word offset
bra.l   10$     ; Longword offset (V4 Core only)
```

PRELIMINARY

Some branch optimising is implemented for backward branches, so that a word offset could be reduced to a byte offset if possible.

3. COMMAND LINE OPTIONS

-b	Produce binary output
-d	Turn off '.' local labels (use with cfcc)
-f	Forward branches default to byte offset
-l	Turn listing on
-o	Produce object code output
-r	Turn on RTS/RTE post NOP generation
-x <val>	Debug Options (see section 4)

4. DEBUG OPTIONS

The debug option (-x) uses a weighted number to turn on one or more print statements:

1	Parser
2	Template matches
4	Expression Evaluation
8	Symbol table lookup/install
16	Forward references
32	Indexed Indirect information
64	Dump all important tables

e.g. -x10 displays template match operation and symbol table info.

PRELIMINARY

5. FILES

<infile>.o	object file output (SREC,OBJ or BIN)
STDOUT	listing and errors

6. IMPLEMENTATION NOTES

This is a classic 2-pass assembler. Pass 1 establishes the symbol table and pass 2 generates the code.

7. BUGS

This assembler is still under development.

1. If the last line of assembler is a comment, output code will not be produced.

2. The assembler may crash if compiling non-ascii files accidentally.

Original Code - Copyright (C) Motorola, used with permission.
Copyright (C) 1998-2000 Defiant Pty Ltd.
All rights reserved.

www.defiant.com.au