RISC Assembler Version 2 User Guide

This document describes the syntax and pseudo-operations of the RISC assembler, version 2.

0) Changes from Version 1

The main changes to the version 1 assembler to do with relocatable expressions (not supported on version 1) and the X/OPEN standard. These include:

- $ is no longer used to precede immediate values; this is to prevent complex expressions from becoming unreadable;
- only one source file can now be specified, and the '-o <object file>' option is now supported;
- relocatable expressions are allowed, with certain syntactical restrictions.
- some bugs have been fixed (no doubt others have been introduced!), and much more error checking is performed.

1) Invocation

The assembler is invoked with:

as [**-g**] [**-o** <object file name>] <source file name>

The optional -g argument causes an assembly listing to be sent to **stdout** (this is included for assembler debugging purposes only). Error messages are included within this listing. The default is to output only erroneous lines and error messages to **stderr**.

The optional -o argument allows an object file name other than the default to be specified. The default object file name consists of a **.o** suffix appended to the source file name (with any **.s** suffix stripped).

The source file must contain RISC assembly language statements in ASCII text form. Source files are assumed to have **.s** extensions; if a name without such an extension is given, the assembler first tries to open a source file with a **.s** extension before trying to open one without.

2) Sections

Assembly output is in COFF format, and consists of three main sections, the **text**, **data** and **bss** sections. Assembly begins by default in the **text** section. The pseudo-operations **text** and **data** toggle the current output section between the **text** and **data** sections. The **bss** section contains only unitialised data; such data is declared with special assembly pseudo-ops, hence it is never necessary to have the current output section be the **bss** section.

If the assembler encounters assembly source which is inappropriate to the current section, a warning is printed and the current section is changed to the appropriate one. If this switch was not performed, relocation entries may be made in the wrong section. To avoid any peculiar effects, the **data** and **text** pseudo-ops should always be used to explicitly change the section, rather than relying on this implicit switch.

3) Syntax

An assembly source program consists of zero or more lines of the form:

[<label>:] [<RISC instruction> [<operands>]] [#<comments>]

or:

[<label>:] [<Pseudo-operation>] [#<comments>]

A **label** is an identifier beginning with a letter, underscore or tilde, and consisting of a sequence of letters, digits, underscores, tildes and percentage signs. Thus:

```
<label>      ::=<identifier>.
<identifier>::=<id start>[<id cont>].
<id start>   ::="a" |...| "z" | "A" |...| "Z" | "_" | "~".
<id cont>    ::=<id start><id cont> | "%"<id cont> | <null>.
<null>       ::=.
```

A tilde is converted to a period in the symbol table.

A **RISC instruction** is one of:

```
<RISC instruction>::=<move> | <arith> | <logic> | <control> | <misc> |
                     <priv> | <stest> | <utest>.
<move>       ::="ldbu" | "ldbs" | "ldsu" | "ldss" | "ldl" | "stb" |
                 "sts" | "stl".
<arith>      ::="add" | "sub" | "subr".
<logic>      ::="and" | "or" | "xor" | "land" | "lor" | "sra" | "sll" |
                 "srl".
<control>    ::="trap" | "jmp" | "jsr" | "bsr" | "bf" | "bt" | "rts" |
                 "rti".
<misc>       ::="ldhi" | "nop".
<priv>       ::="wait" | "ioff" | "ion" | "stpc".
<stest>      ::="teq" | "tne" | "tge" | "tgt" | "tle" | "tlt".
<utest>      ::="ths" | "thi" | "tls" | "tlo".
```

The **operands** of a machine instruction can be of the form:

```
<operands>   ::=[<operand>[,<operand>[,<operand>]]].
<operand>    ::=<null> | <reg> <shf_ind> | <imed> <shf_ind> | <abs>.
<reg>        ::="%r"<decimal integer>.
<shf_ind>    ::=[<shf>]<ind> | [<shf>]<ind> | <null>.
<shf>        ::="*"<data size>.
<data size>  ::= "2" | "4".
<ind>        ::="("<reg>")".
<imed>       ::=<abs>.
<abs>        ::=<ident> | <number>.
<number>     ::=<hex integer> | <decimal integer> | <octal integer>.
```

The only change from version 1 is for <imed>.

Anything following a pound sign # to the end of the containing line is regarded as comments and ignored.

A **pseudo-operation** is one of:

```
<pseudo-operation>    ::=<secop> ¦ <symop> ¦ <dataop>.
<secop>       ::=´text´ ¦ ´data´ ¦ ´org´ <number> ¦ ´align´ <data size>.
<symop>       ::=´set´ <ident>´,´<expr> ¦ ´lcomm <ident>´,´<number> ¦
                  ´comm´ <ident>´,´<number> ¦ ´global´ <ident>.
<dataop>      ::=´byte´ <bexpl> ¦ ´word´ <expl> ¦ ´long´ <expl´ ¦
                  ´space´ <number>.
<bexpl>       ::=<bexp>[,<bexpl>] ¦ <bexp>.
<expl>        ::=<exp>[.<expl>] ¦ <exp>.
<bexp>        ::=<exp> ¦  <char> ¦ <string>.
<exp>         ::=<term>[<addop><exp>].
<term>        ::=<factor>[<mulop><term>].
<factor>      ::=[´-´]<number> ¦ <ident> ¦ ´(´<exp>´)´.
<char>        ::=´´´<character>.
<string>      ::=´"´<char seq>´"´.
<char seq>    ::=<character><char seq> ¦ <null>.
```

Blank lines and whitespace are in general freely allowed. It is also legal to have more than one instruction or pseudo-operation on a line; however, such lines will not be separated in the optional output listing, and may possibly cause error messages to be lost.

4) Pseudo-Operation Summary

4.1) Output Section Control

text selects the **text** section to be the current output section.

data selects the **data** section to be the current output section.

org <number> sets the offset in the current section to the specified number, unless this is less than the current offset.

align <2¦4> causes the offset in the current section to be aligned on the nearest short word or long word boundary.

4.2) Symbol Table Operations

comm <id>,<number> causes <number> bytes of storage to be assigned to <id>. The <id> is made an undefined external. The linker will not allocate space for a **comm** declaration unless it is not allocated anywhere else.

lcomm <id>,<number> causes <number> bytes of storage to be assigned and allocated to <id>. A static entry for the **bss** section is made in the symbol table.

set <id>,<expr> associates the value of the absolute expression with the identifier. This association exists only during assembly; no symbol table entry is actually made for the identifier (more precisely, **set** identifiers exist in a special temporary symbol table). The identifier can subsequently be used as either an immediate or absolute value (unlike any other identifiers, which can never be absolute). The expression can be any simple expression consisting of numbers and identifiers, parentheses, and the operators *, /, +, -, %. No semantic or type checking is performed on expressions.

global <id> makes the <id> externally visible, regardless of its current storage class.

4.3) Data Definition

byte, **word** and **long** are all followed by one or more expressions, separated by commas. Expressions for **byte** may also be characters preceded by a single quote, or strings enclosed in double quotes, but may not be relocatable expressions. Values which exceed the limits for their type are truncated.

space <number> outputs <number> zero bytes.

5) Expressions

Expressions may be of two types, **relocatable** and **absolute**, with the relationship:

<rel expr> ::= <id | number>[<+|->**<abs expr>**]

In other words, a relocatable expression consists of an identifier or number, followed by an optional additive operator and absolute expression. When a relocatable expression is required but an absolute one is given, the absolute expression should be enclosed in parentheses to prevent being interpreted as a relocatable expression.

A relocation entry is only made for a relocatable expression if it begins with an identifier which is not absolute (that is, the identifier is either a relocatable address in one of the sections or is undefined). If this is the case, a value of zero is used for the identifier. Otherwise the value of the absolute identifier or number is used, and the rest of the expression is parsed as an absolute expression.

Absolute expressions can contain relocatable identifiers; these are treated as absolutes and their current values in the symbol table used. A warning is issued when this occurs.

Relocatable expressions are only allowed in three cases: as arguments to **short** or **long** pseudo-operations, and as the destinations of relative branches. All other expressions are treated as absolutes.

6) RISC Instruction Set Details

6.1) Operand Types

The following operand types are supported:

Non-Terminal	Type	Syntax
<reg>	Register	%rn
<ireg>	Register Indirect with Offset	nn(%rn) or id(%rn)
<sireg>	Register Indirect with Shifted Offset	nn*sh(%rn)
<rreg>	Register Indirect with Register Offset	%rn(%rm)
<srreg>	Register Indirect with Shifted Register Offset	%rn*sh(%rm)
<imed>	Immediate	nnnn or id or <relad>
<relad>	Relocatable Address	see section 5
<address>	Any Address	<ireg¦rreg¦sireg¦srreg>
<arith>	Arithmetic Operand	<reg¦imed>

6.2) Load/Store Instructions

Instruction	Name
ldbu <address>,<reg>	Load byte unsigned
ldbs <address>,<reg>	Load byte signed
ldsu <address>,<reg>	Load short unsigned
ldss <address>,<reg>	Load short signed
ldl <address>,<reg>	Load long
stb <reg>,<address>	Store byte
sts <reg>,<address>	Store short
stl <reg>,<address.	Store long

6.3) Arithmetic Instructions

Instruction	Name
add <reg>,<arith>,<reg>	Add
sub <reg>,<arith>,<reg>	Subtract
subr <reg>,<imed> ,<reg>	Subtract reverse

6.4) Logical Instructions

Instruction	Name
and <reg>,<arith>,<reg>	Bitwise AND
or <reg>,<arith>,<reg>	Bitwise OR
xor <reg>,<arith>,<reg>	Bitwise EXCLUSIVE-OR
land <reg>,<arith>,<reg>	Logical AND
lor <reg>,<arith>,<reg>	Logical OR
sll <reg>,<arith>,<reg>	Shift left logical
srl <reg>,<arith>,<reg>	Shift right logical

6.5) Control Instructions

Instruction	Name
trap <reg>,<imed>	Trap to service routine
jmp <imed>	Jump absolute
jsr <reg>,<imed>	Jump to absolute subroutine
bf <reg>,<relad>	Branch relative on false
bt <reg>,<relad>	Branch relative on true
bsr <reg>,<relad>	Branch relative to subroutine
rts <reg>	Return from subroutine
rti <reg>	Return from subroutine & enable interrupts

6.5) Miscellaneous

Instruction	Name
ldhi <reg>,<imed>	Load high immediate
nop	No operation

6.6) Privileged

Instruction	Name
wait	Wait on interrupt
ioff	Turn interrupts off
ion	Turn interrupts on
stpc <reg>	Store program counter

6.7) Signed Tests

Instruction	Name
teq <reg>,<arith>,<reg>	Test equal
tne <reg>,<arith>,<reg>	Test not equal
tge <reg>,<arith>,<reg>	Test greater than or equal
tgt <reg>,<arith>,<reg>	Test greater than
tle <reg>,<arith>,<reg>	Test less than or equal
tlt <reg>,<arith>,<reg>	Test less than

6.8) Unsigned Test

Instruction	Name
ths <reg>,<arith>,<reg>	Test higher than or same
thi <reg>,<arith>,<reg>	Test higher than
tls <reg>,<arith>,<reg>	Test lower than or same
tlo <reg>,<arith>,<reg>	Test lower than

7) Error Messages

Illegal attempt to redefine <identifier>
A previously defined identifier has been redefined.

Label <identifier> used but not defined
An identifier has been used as a label but has never been declared.

<token> expected
The lexical analyser expected a token other than the one encountered.

Unrecognised instruction <string>
The lexical analyser was expecting a RISC or assembler instruction, but did not recognise the token it found.

Numeric constant expected
Immediate value expected
Register number expected
These are all similar; produced by the lexical analyser when it expected a specific token type and found some other type.

Invalid register number <number>
The number encountered after a '%r' prefix does not specify a valid RISC register

Invalid shift size <number> for this instruction
The shift field used is the wrong size for the instruction.

Immediate value exceeds allowed range
A number was found whose magnitude is too large for its type.

New origin is less than old
An attempt was made to use **org** to decrease the current section offset from its present value.

Bad expression
An error occurred while parsing an expression.

Unknown identifier <identifier> in expression
An undefined identifier was used in an absolute expression.

Invalid or bad argument type <arg>
An inappropriate or erroneous operand was used in an instruction.

Align value must be 2 or 4
A value other than 2 or 4 was used as the argument to **align**.

Destination <label> must be relocatable
The destination of a relative branch instruction was not a relocatable label.

Unexpected end-of-file
The end of the source file was reached while parsing an instruction.

8) Warnings (non-fatal errors)

Switching output to text section
A RISC instruction was found while in the data section.

Switching output to data section
A **byte**, **word** or **long** pseudo-operation was found while in the text section.

Identifier <identifier> should be absolute
A relocatable identifier was found when an absolute value was expected.

Link Editor Specification

Syntax: ld [options] file ...

Input:

Options:

-e epsym	Set the default entry point address for the output file to be epsym
-lxxx	Search library /usr/lib/libxxx
-o outfile	Rename output file
-r	Don't strip reloc entries (output file not executable, unresolved references allowed)
-s	Strip symbolic info from output (default)
-u symname	Enter symname as undefined in sym.table.
-L dir	Look in dir instead of /usr/lib
-V	Return version

Files:

Files can be object files produced by the assembler, outputs of previous ld runs, or archive libraries.

Processing:

Relocation Types:

The following relocation types are supported. Note that the sizes long, word, etc actually have shorter sizes in the text section - word for jsr (14 bits) and long for all others (19 bits).

* does ass do this?

SHORT RELATIVE
WORD RELATIVE
LONG RELATIVE
LONG PC RELATIVE
ABSOLUTE

Archive Libraries:

Libraries are searched at the point they are encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The library (archive) symbol table is searched to resolve external references that can be satisfied by library members. The ordering of library members is unimportant, unless there exist multiple library members defining the same external symbol.
ld searches the archive symbol table as many times as needed to resolve all references. ld reads each the appropriate object files from the archive, relocates it, adds it to the output file together with its stab.

Other files:

Object files are combined into a single file, relocation is performed and external symbols are resolved. If any input file is not an object file, it is assumed to be a library.
When an object file is found, ld reads the file, relocates the code, and adds this onto the end of the output file. The object file's stab is then merged with the output file's.

General:

Once all the command line arguments have been processed (link edit) the load can be done. The symbol table is sorted, and all unresolved references are resolved. Names that remain undefined after the load cause error messages and no output.

Output: Finally sort stab into debug order.

The output of ld goes by default to a.out, which is executable if no errors occurred during the load. The symbols etext, edata and end as seen from a C program are reserved and defined by the link editor.

GRATTAN

Instruction Bit Patterns

General Form.

The instruction all fall within the following formats:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
G	G	0	0	0	I	A	A	Rd				Rs				Rs'				Cond				8-bit immed							
								Rd				Rs				16-bit immed															
								Rd				20-bit immed.																			
								Rd				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	5-bit immed			

Op Field.

The Op Field.

G	G	0	0	0	I	A	A

The components of this field are:

G	G

Instruction Group: 0 0 - Load / Store Instructions
0 1 - Arithmetic Instructions
1 0 - Control Flow Instructions
1 1 - Miscellaneous (actually 2 groups)

0	0	0

Operation: Specifies which operation in the group to perform.

I

Immediate Bit: Specifies whether an immediate operand is present (Can be regarded as an extension of 000).

A	A

Addressing Mode: This works in conjunction with the I bit

Instruction Group | 0 0 | - The Load & Store Instructions.

These instructions make use of all the addressing bits, as follows:

I	A	A	Operands (Loads)	Operands (Stores)
0	0	0	Rs' (Rs), Rd	Rd, Rs' (Rs)
1	0	0	Imm (Rs), Rd	Rd, Imm (Rs)
0	0	1	Rs*Size(Rs), Rd	Rd, Rs'*Size(Rs)
0	1	0	(-Rs), Rd	Rd, (-Rs)
0	1	1	(Rs)+, Rd	Rd, (Rs)+

The Operation components of the Op Field are

O	O	O	Operation	Mnemonic
0	0	0	Load byte unsigned	ldbu
0	0	1	Load byte signed	ldbs
0	1	0	Load short unsigned	ldsu
0	1	1	Load short signed	ldss
1	0	0	Load Long	ldl
1	0	1	Store byte	stb
1	1	0	Store short	sts
1	1	1	Store long	stl

Examples of Load Short Signed. (X = don't care)

0	0	0	1	1	0	0	0	Rd	Rs	Rs'	X X X X X X X X X X X X	ldss Rs'(Rs), Rd
0	0	0	1	1	1	0	0	Rd	Rs	16-bit immediate value		ldss Imm(Rs), Rd
0	0	0	1	1	0	0	1	Rd	Rs	Rs'	X X X X X X X X X X X X	ldss Rs'*2(Rs), Rd
0	0	0	1	1	0	1	0	Rd	Rs	X X X X X X X X X X X X X X X X		ldss (-Rs), Rd
0	0	0	1	1	0	1	1	Rd	Rs	X X X X X X X X X X X X X X X X		ldss (Rs)+, Rd
G	G	O	O	O	I	A	A	Rd	Rs			

Instruction Group [01] - The Arithmetic Instructions

The Operation Components of the Op Field are

[000]	Operation	Mnemonic	Possible Operand Combinations	
000	32-bit Add	add	Rs, Rs', Rd	Rs, Imm, Rd
001	32-bit Subtract	sub	Rs, Rs', Rd	Rs, Imm, Rd
010	32-bit Bitwise AND	and	Rs, Rs', Rd	Rs, Imm, Rd
011	32-bit Bitwise OR	or	Rs, Rs', Rd	Rs, Imm, Rd
100	32-bit Bitwise XOR	xor	Rs, Rs', Rd	Rs, Imm, Rd
101	Logical AND	land	Rs, Rs', Rd	Rs, Imm, Rd
110	Logical OR	lor	Rs, Rs', Rd	Rs, Imm, Rd
111	32-bit Reverse Subtract	subr	(if I = 1)	Rs, Imm, Rd
111	~~Logical NOT~~ (eq %r0, rs, rd)	lnot	(if I = 0)	Rs, Rd

lnot only has the form: lnot Rs, Rd (I = 0)
subr only has the form: subr Rs, Imm, Rd (I = 1)
All other operations have two forms: op Rs, Rs', Rd (I = 0)
op Rs, Imm, Rd. (I = 1).

We thus use the following 'address encoding':

01	000	0	XX	Rd	Rs	Rs'	X	X	X	X	X	X	X	X	X	X	X	X	add Rs, Rs', Rd
01	001	1	XX	Rd	Rs	16-bit immediate value																		or Rs, Imm, Rd
01	111	0	XX	Rd	Rs	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	lnot Rs, Rd
01	111	1	XX	Rd	Rs	16-bit immediate value																		subr Rs, Imm, Rd
GG	000	I	AA	Rd	Rs																			

Instruction Group 10 - Control Flow Instructions.

These have the following Operation Components of the Op Field:

0 0 0	Operation	Mnemonic	Possible Operand Combinations	
0 0 0	Trap to Routine	trap	Rd, Imm	
0 0 1	Jump	jmp	Rd, Rs'(Rs)	Rd, Imm(Rs)
0 1 0	Jump Subroutine	jsr	Rd, Rs'(Rs)	Rd, Imm(Rs)
0 1 1	Branch Subroutine	bsr	Rd, Imm	
1 0 0	Branch if False	bf	Rd, Imm	
1 0 1	Branch if True	bt	Rd, Imm	
1 1 0	Return from Subroutine	rts	Rd	
1 1 1	Return from Subroutine and Enable Interrupts	rti	Rd	

We thus have the following possibilities:

10	000	X	XX	Rd	XXXXXXXXXXXX	8-bit trap number			trap Rd, Imm
10	001	0	XX	Rd	Rs	Rs'	XXXXXXXXXXXX		jmp Rd, Rs'(Rs)
10	001	1	XX	Rd	Rs	16-bit immediate value			jmp Rd, Imm(Rs)
10	010	0	XX	Rd	Rs	Rs'	XXXXXXXXXXXX		jsr Rd, Rs'(Rs)
10	010	1	XX	Rd	Rs	16-bit immediate value			jsr Rd, Imm(Rs)
10	011	X	XX	Rd	20-bit immediate value				bsr Rd, Imm
10	100	X	XX	Rd	20-bit immediate value				bf Rd, Imm
10	101	X	XX	Rd	20-bit immediate value				bt Rd, Imm
10	110	X	XX	Rd	XXXXXXXXXXXXXXXXXXXX				rts Rd
10	111	X	XX	Rd	XXXXXXXXXXXXXXXXXXXX				rti Rd
GG	000	I	AA	Rd	Rs				