occam-2 compiler library specification

Conor O’Neill and Stephen Clarke

SW-0063-5

INMOS Limited Confidential

APPROVED 4 March, 1991

Contents

1 Change history

1.1 Changes since issue of SW-0063-4.

- Changed names of routines to end in **%O** or **%CHK**.

1.2 Changes since issue of SW-0063-3.

- **REAL** to **INT16** conversion routines only required for 16-bit.
- Added predefines **IEEE32REM** and **IEEE64REM**.

1.3 Changes since issue of SW-0063-2 of February 1, 1990

- The compiler libraries have been coalesced from nine into three.
- Added TA and TB tables in the summary.
- TA version of **NOTFINITE** is compiled inline.

1.4 Changes since issue of January 24, 1990

- Compiler libraries are now provided in UNIVERSAL mode.

2 Introduction

This document lists all the library routines which may be called implicitly by code compiled by the Inmos occam compiler, and the conditions in which they will be called. It also lists the predefined routines which must be supplied in library form.

The occam compiler can compile in HALT, STOP or UNIVERSAL error mode; therefore it may expect to find these routines in any of these error modes in the compiler libraries.

The table in section 6 details which processors each routine should be available for.

The occam compiler needs to read a compiler library when it has to compile a call into that compiler library, as it needs to know the workspace and vector space requirements of the called routine. For efficiency reasons, the compiler insists that the library routines do not use vectorspace, hence a vectorspace parameter is never needed. Theoretically, the compiler should also check that the formal parameters are of the correct types. This is not done at present.

The occam compiler assumes the following names for the compiler libraries:

	Target processor		
	T212	T414	T800
	T222	T425	T801
	T225	T400	T805
Error mode	M212	TA / TB	
Any	occam2.lib	occama.lib	occam8.lib

A *predefined* routine is a routine which is automatically defined by the compiler before the first line of user source. These may be descoped in the same way as any other identifier. Predefined routines are in some cases translated into inline code sequences and in other cases into calls into the compiler libraries. For the sake of completeness, this document lists *all* predefined routine names recognised by the compiler: those predefined routines which are compiled inline for particular target processors will not appear in the compiler library for that processor. Those predefined routines which are *always* compiled inline will not appear in any compiler library.

This document only details the entry points of the compiler libraries as required by the compiler: any routines which are internal to the compiler libraries are not listed. These are currently implemented in another library `occamutl.lib` which holds routines for all processor types and error modes.

3 Operations

A routine of the form

type op **%CHK**

performs the occam operation *op* upon operands of occam type *type*, where *type* is one of **INT16**, **INT32**, **INT64**, **REAL32** or **REAL64**, and *op* is one of **ADD**, **SUB**, **MUL**, **DIV**, **REM**, **PLUS**, **MINUS**, **TIMES**, **BITAND**, **BITOR**, **XOR**, **BITNOT**, **LSHIFT**, **RSHIFT**, **XOR**, **GT** or **EQ**.

For example, **INT32PLUS%CHK** performs the occam **PLUS** operation upon two operands of type **INT32**.

The routines **REAL32OPERR%CHK** and **REAL64OPERR%CHK** implement the occam operations **+**, **-**, ***** and **/** upon **REAL32** and **REAL64** operands respectively. The **Op** parameter takes the following values:

Value	Meaning	
0	Add	(occam **+**)
1	Subtract	(occam **-**)
2	Multiply	(occam *****)
3	Divide	(occam **/**)

The routines **REAL32REMERR%CHK** and **REAL64REMERR%CHK** implement the occam remainder operation upon **REAL32** and **REAL64** operands respectively. They are equivalent to **REAL32OP**, **REAL64OP**, **REAL32REM** and **REAL64REM** (see "occam 2 Reference Manual", Prentice-Hall 1988) except that they perform occam error checking for errors and overflows, rather than the IEEE method of using infinities and Not-a-Numbers.

The routines **REAL32EQERR%CHK**, **REAL64EQERR%CHK**, **REAL32GTERR%CHK**, and **REAL64GTERR%CHK** perform the occam operations **=** and **>** upon **REAL32** and **REAL64** operands. They are equivalent to **REAL32EQ**, **REAL64EQ**, **REAL32GT** and **REAL64GT** (see "occam 2 Reference Manual", Prentice-Hall 1988) except that they perform occam error checking for errors and overflows, rather than the IEEE method of using infinities and Not-a-Numbers.

3.1 INT16 operations

- **INT16 FUNCTION INT16ADD%CHK (VAL INT16 A, VAL INT16 B)**
  This routine is called to perform **+** upon two **INT16**s on a 32-bit processor.
- **INT16 FUNCTION INT16SUB%CHK (VAL INT16 A, VAL INT16 B)**
  This routine is called to perform **-** upon two **INT16**s on a 32-bit processor.
- **INT16 FUNCTION INT16MUL%CHK (VAL INT16 A, VAL INT16 B)**
  This routine is called to perform ***** upon two **INT16**s on a 32-bit processor.
- **INT16 FUNCTION INT16DIV%CHK (VAL INT16 A, VAL INT16 B)**
  This routine is called to perform **/** upon two **INT16**s on a 32-bit processor.
- **INT16 FUNCTION INT16REM%CHK (VAL INT16 A, VAL INT16 B)**
  This routine is called to perform **REM** upon two **INT16**s on a 32-bit processor.
- **INT16 FUNCTION INT16PLUS%CHK (VAL INT16 A, VAL INT16 B)**
  This routine is called to perform **PLUS** upon two **INT16**s on a 32-bit processor.

- `INT16 FUNCTION INT16MINUS%CHK (VAL INT16 A, VAL INT16 B)`
  This routine is called to perform `MINUS` upon two `INT16`s on a 32-bit processor.
- `INT16 FUNCTION INT16TIMES%CHK (VAL INT16 A, VAL INT16 B)`
  This routine is called to perform `TIMES` upon two `INT16`s on a 32-bit processor.
- `INT16 FUNCTION INT16BITAND%CHK (VAL INT16 A, VAL INT16 B)`
  This routine is called to perform `BITAND` upon two `INT16`s on a 32-bit processor.
- `INT16 FUNCTION INT16BITOR%CHK (VAL INT16 A, VAL INT16 B)`
  This routine is called to perform `BITOR` upon two `INT16`s on a 32-bit processor.
- `INT16 FUNCTION INT16XOR%CHK (VAL INT16 A, VAL INT16 B)`
  This routine is called to perform `><` upon two `INT16`s on a 32-bit processor.
- `INT16 FUNCTION INT16LSHIFT%CHK (VAL INT16 A, VAL INT B)`
  This routine is called to perform `<<` upon an `INT16` on a 32-bit processor.
- `INT16 FUNCTION INT16RSHIFT%CHK (VAL INT16 A, VAL INT B)`
  This routine is called to perform `>>` upon an `INT16` on a 32-bit processor.
- `INT16 FUNCTION INT16BITNOT%CHK (VAL INT16 A)`
  This routine is called to perform `BITNOT` upon an `INT16` on a 32-bit processor.
- `BOOL FUNCTION INT16GT%CHK (VAL INT16 A, VAL INT16 B)`
  This routine is called to perform `<`, `<=`, `>=` and `>` upon two `INT16`s on a 32-bit processor.
- `BOOL FUNCTION INT16EQ%CHK (VAL INT16 A, VAL INT16 B)`
  This routine is called to perform `=`, and `<>` upon two `INT16`s on a 32-bit processor.

3.2 `INT32` operations

- `INT32 FUNCTION INT32MUL%CHK (VAL INT32 A, VAL INT32 B)`
  This routine is called to perform `*` upon two `INT32`s on a 16-bit processor.
- `INT32 FUNCTION INT32DIV%CHK (VAL INT32 Dvd, VAL INT32 Dvsr)`
  This routine is called to perform `/` upon two `INT32`s on a 16-bit processor.
- `INT32 FUNCTION INT32REM%CHK (VAL INT32 Dvd, VAL INT32 Dvsr)`
  This routine is called to perform `REM` upon two `INT32`s on a 16-bit processor.

3.3 `INT64` operations

- `INT64 FUNCTION INT64ADD%CHK (VAL INT64 A, VAL INT64 B)`
  This routine is called to perform `+` upon two `INT64`s on a 16-bit processor.
- `INT64 FUNCTION INT64SUB%CHK (VAL INT64 A, VAL INT64 B)`
  This routine is called to perform `-` upon two `INT64`s on a 16-bit processor.
- `INT64 FUNCTION INT64MUL%CHK (VAL INT64 U, VAL INT64 V)`
  This routine is called to perform `*` upon two `INT64`s on all processors.
- `INT64 FUNCTION INT64DIV%CHK (VAL INT64 U, VAL INT64 V)`
  This routine is called to perform `/` upon two `INT64`s on all processors.

- **`INT64 FUNCTION INT64REM%CHK (VAL INT64 U, VAL INT64 V)`**
  This routine is called to perform **`REM`** upon two **`INT64`**s on all processors.

- **`INT64 FUNCTION INT64PLUS%CHK (VAL INT64 A, VAL INT64 B)`**
  This routine is called to perform **`PLUS`** upon two **`INT64`**s on a 16-bit processor.

- **`INT64 FUNCTION INT64MINUS%CHK (VAL INT64 A, VAL INT64 B)`**
  This routine is called to perform **`MINUS`** upon two **`INT64`**s on a 16-bit processor.

- **`INT64 FUNCTION INT64TIMES%CHK (VAL INT64 U, VAL INT64 V)`**
  This routine is called to perform **`TIMES`** upon two **`INT64`**s on a 16-bit processor.

- **`INT64 FUNCTION INT64BITAND%CHK (VAL INT64 U, VAL INT64 V)`**
  This routine is called to perform **`BITAND`** upon two **`INT64`**s on a 16-bit processor.

- **`INT64 FUNCTION INT64BITOR%CHK (VAL INT64 U, VAL INT64 V)`**
  This routine is called to perform **`BITOR`** upon two **`INT64`**s on a 16-bit processor.

- **`INT64 FUNCTION INT64XOR%CHK (VAL INT64 U, VAL INT64 V)`**
  This routine is called to perform `><` upon two **`INT64`**s on a 16-bit processor.

- **`INT64 FUNCTION INT64LSHIFT%CHK (VAL INT64 B, VAL INT Places)`**
  This routine is called to perform `<<` upon an **`INT64`** on a 16-bit processor.

- **`INT64 FUNCTION INT64RSHIFT%CHK (VAL INT64 B, VAL INT Places)`**
  This routine is called to perform `>>` upon an **`INT64`** on a 16-bit processor.

- **`INT64 FUNCTION INT64BITNOT%CHK (VAL INT64 B)`**
  This routine is called to perform **`BITNOT`** upon an **`INT64`** on a 16-bit processor.

- **`BOOL FUNCTION INT64GT%CHK (VAL INT64 U, VAL INT64 V)`**
  This routine is called to perform `<=`, `<`, `>=` and `>` upon two **`INT64`**s on a 16-bit processor.

- **`BOOL FUNCTION INT64EQ%CHK (VAL INT64 U, VAL INT64 V)`**
  This routine is called to perform `=` and `<>` upon two **`INT64`**s on a 16-bit processor.

3.4 `REAL32` operations

- **`REAL32 FUNCTION REAL32OPERR%CHK (VAL REAL32 X,VAL INT Op,VAL REAL32 Y)`**

  This routine is called to perform `+`, `-`, `*` and `/` upon two **`REAL32`**s on a processor with no floating-point unit.

- **`REAL32 FUNCTION REAL32REMERR%CHK (VAL REAL32 X,VAL REAL32 Y)`**
  This routine is called to perform **`REM`** upon two **`REAL32`**s on a processor with no floating-point unit.

- **`BOOL FUNCTION REAL32GTERR%CHK (VAL REAL32 X,VAL REAL32 Y)`**
  This routine is called to perform `<`, `<=`, `>=` and `>` upon two **`REAL32`**s on a processor with no floating-point unit.

- **`BOOL FUNCTION REAL32EQERR%CHK (VAL REAL32 X,VAL REAL32 Y)`**
  This routine is called to perform `=`, and `<>` upon two **`REAL32`**s on a processor with no floating-point unit.

3.5 `REAL64` operations

- **`REAL64 FUNCTION REAL64OPERR%CHK (VAL REAL64 X,VAL INT Op,VAL REAL64 Y)`**

  This routine is called to perform **`+`**, **`-`**, **`*`** and **`/`** upon two **`REAL64`**s on a processor with no floating-point unit.

- **`REAL64 FUNCTION REAL64REMERR%CHK (VAL REAL64 X,VAL REAL64 Y)`**
  This routine is called to perform **`REM`** upon two **`REAL64`**s on a processor with no floating-point unit.

- **`BOOL FUNCTION REAL64GTERR%CHK (VAL REAL64 X,VAL REAL64 Y)`**
  This routine is called to perform **`<`**, **`<=`**, **`>=`** and **`>`** upon two **`REAL64`**s on a processor with no floating-point unit.

- **`BOOL FUNCTION REAL64EQERR%CHK (VAL REAL64 X,VAL REAL64 Y)`**
  This routine is called to perform **`=`**, and **`<>`** upon two **`REAL64`**s on a processor with no floating-point unit.

4 Type conversions

A routine of the form

type1 **TO** *type2* **%CHK**

performs the occam type conversion from *type1* to *type2*, where *type1* and *type2* are one of **INT16**, **INT32**, **INT64**, **REAL32** or **REAL64**. **REAL** type conversions take a rounding mode if the conversion cannot be performed exactly, which may take the values:

Mode	Meaning	
0	Round to zero	(occam **TRUNC**)
1	Round to nearest	(occam **ROUND**)

4.1 INT16 conversions

- **INT64 FUNCTION INT16TOINT64%CHK (VAL INT16 B)**
  This routine is called to type convert an **INT16** to an **INT64** on a 16-bit processor.

- **REAL32 FUNCTION INT16TOREAL32%CHK (VAL INT16 N)**
  This routine is called to type convert an **INT16** to a **REAL32** on a 16-bit processor.

- **REAL64 FUNCTION INT16TOREAL64%CHK (VAL INT16 N)**
  This routine is called to type convert an **INT16** to a **REAL64** on a 16-bit processor.

4.2 INT32 conversions

- **INT64 FUNCTION INT32TOINT64%CHK (VAL INT32 B)**
  This routine is called to type convert an **INT32** to an **INT64** on a 16-bit processor.

- **REAL32 FUNCTION INT32TOREAL32%CHK (VAL INT Mode,VAL INT32 N)**
  This routine is called to type convert an **INT32** to a **REAL32** on a processor with no floating-point unit. The rounding mode will be round to nearest if a **ROUND** conversion is specified, or round to zero if a **TRUNC** conversion is specified.

- **REAL64 FUNCTION INT32TOREAL64%CHK (VAL INT32 N)**
  This routine is called to type convert an **INT32** to a **REAL64** on a processor with no floating-point unit.

4.3 INT64 conversions

- **INT16 FUNCTION INT64TOINT16%CHK (VAL INT64 B)**
  This routine is called to type convert an **INT64** to an **INT16** on a 16-bit processor.

- **INT32 FUNCTION INT64TOINT32%CHK (VAL INT64 B)**
  This routine is called to type convert an **INT64** to an **INT32** on a 16-bit processor.

- **REAL32 FUNCTION INT64TOREAL32%CHK (VAL INT Mode,VAL INT64 N)**
  This routine is called to type convert an **INT64** to a **REAL32** on a processor with no floating-point unit. The rounding mode will be round to nearest if a **ROUND** conversion is specified, or round to zero if a **TRUNC** conversion is specified.

- **`REAL64 FUNCTION INT64TOREAL64%CHK (VAL INT Mode,VAL INT64 N)`**
  This routine is called to type convert an **`INT64`** to a **`REAL64`** on a processor with no floating-point unit. The rounding mode will be round to nearest if a **`ROUND`** conversion is specified, or round to zero if a **`TRUNC`** conversion is specified.

4.4 `REAL32` conversions

- **`INT16 FUNCTION REAL32TOINT16%CHK (VAL INT Mode,VAL REAL32 X)`**
  This routine is called to type convert a **`REAL32`** to an **`INT16`** on a 16-bit processor. The rounding mode will be round to nearest if a **`ROUND`** conversion is specified, or round to zero if a **`TRUNC`** conversion is specified.

- **`INT32 FUNCTION REAL32TOINT32%CHK (VAL INT Mode,VAL REAL32 X)`**
  This routine is called to type convert a **`REAL32`** to an **`INT32`** on a processor with no floating-point unit. The rounding mode will be round to nearest if a **`ROUND`** conversion is specified, or round to zero if a **`TRUNC`** conversion is specified.

- **`INT64 FUNCTION REAL32TOINT64%CHK (VAL INT Mode,VAL REAL32 X)`**
  This routine is called to type convert a **`REAL32`** to an **`INT64`** on a processor with no floating-point unit. The rounding mode will be round to nearest if a **`ROUND`** conversion is specified, or round to zero if a **`TRUNC`** conversion is specified.

- **`REAL64 FUNCTION REAL32TOREAL64%CHK (VAL REAL32 X)`**
  This routine is called to type convert a **`REAL32`** to a **`REAL64`** on a processor with no floating-point unit.

4.5 `REAL64` conversions

- **`INT16 FUNCTION REAL64TOINT16%CHK (VAL INT Mode,VAL REAL64 X)`**
  This routine is called to type convert a **`REAL64`** to an **`INT16`** on a 16-bit processor. The rounding mode will be round to nearest if a **`ROUND`** conversion is specified, or round to zero if a **`TRUNC`** conversion is specified.

- **`INT32 FUNCTION REAL64TOINT32%CHK (VAL INT Mode,VAL REAL64 X)`**
  This routine is called to type convert a **`REAL64`** to an **`INT32`** on a processor with no floating-point unit. The rounding mode will be round to nearest if a **`ROUND`** conversion is specified, or round to zero if a **`TRUNC`** conversion is specified.

- **`INT64 FUNCTION REAL64TOINT64%CHK (VAL INT Mode,VAL REAL64 X)`**
  This routine is called to type convert a **`REAL64`** to an **`INT64`** on a processor with no floating-point unit. The rounding mode will be round to nearest if a **`ROUND`** conversion is specified, or round to zero if a **`TRUNC`** conversion is specified.

- **`REAL32 FUNCTION REAL64TOREAL32%CHK (VAL INT Mode,VAL REAL64 X)`**
  This routine is called to type convert a **`REAL64`** to a **`REAL32`** on a processor with no floating-point unit. The rounding mode will be round to nearest if a **`ROUND`** conversion is specified, or round to zero if a **`TRUNC`** conversion is specified.

5 Predefined routines

The occam compiler automatically recognises a large number of predefined routines, which are in some cases translated into inline code sequences and in other cases into calls into the compiler libraries. Note that all the compiler libraries have entry point names which are formed by adding `'%O'` to the end of the predefine name. This section lists all of these names, and describes when they will be generated inline, and when they will be calls into the compiler libraries. The actual code which is generated for the inline expansions is described in SW-0078 ("Code generated for predefined routines").

All predefined routines named here are specified in SW-0044 ("occam 2 Language Implementation Manual") or "occam 2 Reference Manual", Prentice-Hall 1988.

5.1 Multiple-length integer arithmetic functions

- **`INT FUNCTION LONGADD (VAL INT left, right, carry.in)`**
  Always compiled inline.

- **`INT FUNCTION LONGSUM (VAL INT left, right, carry.in)`**
  Always compiled inline.

- **`INT FUNCTION LONGSUB (VAL INT left, right, borrow.in)`**
  Always compiled inline.

- **`INT, INT FUNCTION LONGDIFF (VAL INT left, right, borrow.in)`**
  Always compiled inline.

- **`INT, INT FUNCTION LONGPROD (VAL INT left, right, carry.in)`**
  Always compiled inline.

- **`INT, INT FUNCTION LONGDIV (VAL INT dvd.hi, dvd.lo, dvsr)`**
  Always compiled inline.

- **`INT, INT FUNCTION SHIFTRIGHT (VAL INT hi.in, lo.in, places)`**
  Always compiled inline.

- **`INT, INT FUNCTION SHIFTLEFT (VAL INT hi.in, lo.in, places)`**
  Always compiled inline.

- **`INT, INT, INT FUNCTION NORMALISE (VAL INT hi.in, lo.in)`**
  Always compiled inline.

- **`INT FUNCTION ASHIFTRIGHT (VAL INT argument, places)`**
  Always compiled inline.

- **`INT FUNCTION ASHIFTLEFT (VAL INT argument, places)`**
  Always compiled inline.

- **`INT FUNCTION ROTATERIGHT (VAL INT argument, places)`**
  Always compiled inline.

- **`INT FUNCTION ROTATELEFT (VAL INT argument, places)`**
  Always compiled inline.

5.2 Floating point functions

- **`REAL32 FUNCTION ABS (VAL REAL32 X)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`BOOL FUNCTION ISNAN (VAL REAL32 X)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`BOOL FUNCTION NOTFINITE (VAL REAL32 X)`**
  Compiled inline on a 32-bit processor, otherwise compiled as a library call. Uses the floating point unit, or the floating point support instructions, if they are available.

- **`BOOL FUNCTION ORDERED (VAL REAL32 X, Y)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`REAL32 FUNCTION MULBY2 (VAL REAL32 X)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`REAL32 FUNCTION DIVBY2 (VAL REAL32 X)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`REAL32 FUNCTION SQRT (VAL REAL32 X)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`REAL32 FUNCTION FPINT (VAL REAL32 X)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`REAL32 FUNCTION MINUSX (VAL REAL32 X)`**
  Compiled inline on a 32-bit processor, otherwise compiled as a library call.

- **`REAL64 FUNCTION DABS (VAL REAL64 X)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`BOOL FUNCTION DISNAN (VAL REAL64 X)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`BOOL FUNCTION DNOTFINITE (VAL REAL64 X)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`BOOL FUNCTION DORDERED (VAL REAL64 X, Y)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- **`REAL64 FUNCTION DMULBY2 (VAL REAL64 X)`**
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.

- `REAL64 FUNCTION DDIVBY2 (VAL REAL64 X)`
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.
- `REAL64 FUNCTION DSQRT (VAL REAL64 X)`
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.
- `REAL64 FUNCTION DFPINT (VAL REAL64 X)`
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.
- `REAL64 FUNCTION DMINUSX (VAL REAL64 X)`
  Compiled inline on a processor which has a floating-point unit, otherwise compiled as a library call.
- `REAL32 FUNCTION SCALEB (VAL REAL32 X, VAL INT n)`
  Always compiled as a library call.
- `REAL64 FUNCTION DSCALEB (VAL REAL64 X, VAL INT n)`
  Always compiled as a library call.
- `REAL32 FUNCTION COPYSIGN (VAL REAL32 X, Y)`
  Always compiled as a library call.
- `REAL64 FUNCTION DCOPYSIGN (VAL REAL64 X, Y)`
  Always compiled as a library call.
- `REAL32 FUNCTION NEXTAFTER (VAL REAL32 X, Y)`
  Always compiled as a library call.
- `REAL64 FUNCTION DNEXTAFTER (VAL REAL64 X, Y)`
  Always compiled as a library call.
- `REAL32 FUNCTION LOGB (VAL REAL32 X)`
  Always compiled as a library call.
- `REAL64 FUNCTION DLOGB (VAL REAL64 X)`
  Always compiled as a library call.
- `INT, REAL32 FUNCTION FLOATING.UNPACK (VAL REAL32 X)`
  Always compiled as a library call.
- `INT, REAL64 FUNCTION DFLOATING.UNPACK (VAL REAL64 X)`
  Always compiled as a library call.
- `BOOL, INT32, REAL32 FUNCTION ARGUMENT.REDUCE (VAL REAL32 X, Y, Y.err)`
  Always compiled as a library call.
- `BOOL, INT32, REAL64 FUNCTION DARGUMENT.REDUCE (VAL REAL64 X, Y, Y.err)`

  Always compiled as a library call.

5.3 Full IEEE arithmetic functions

- `REAL32 FUNCTION REAL32OP (VAL REAL32 X, VAL INT Op, VAL REAL32 Y)`

Always compiled as a library call.

- `REAL64 FUNCTION REAL64OP (VAL REAL64 X, VAL INT Op, VAL REAL64 Y)`
  Always compiled as a library call.

-
  ```
  BOOL, REAL32 FUNCTION IEEE32OP (VAL REAL32 X, VAL INT Rm, Op,
                                  VAL REAL32 Y)
  ```
  Always compiled as a library call.

-
  ```
  BOOL, REAL64 FUNCTION IEEE64OP (VAL REAL64 X, VAL INT Rm, Op,
                                  VAL REAL64 Y)
  ```
  Always compiled as a library call.

- `REAL32 FUNCTION REAL32REM (VAL REAL32 X, VAL REAL32 Y)`
  Always compiled as a library call.

- `REAL64 FUNCTION REAL64REM (VAL REAL64 X, VAL REAL64 Y)`
  Always compiled as a library call.

- `BOOL, REAL32 FUNCTION IEEE32REM (VAL REAL32 X, VAL REAL32 Y)`
  Always compiled as a library call.

- `BOOL, REAL64 FUNCTION IEEE64REM (VAL REAL64 X, VAL REAL64 Y)`
  Always compiled as a library call.

- `BOOL FUNCTION REAL32EQ (VAL REAL32 X, Y)`
  Always compiled as a library call.

- `BOOL FUNCTION REAL64EQ (VAL REAL64 X, Y)`
  Always compiled as a library call.

- `BOOL FUNCTION REAL32GT (VAL REAL32 X, Y)`
  Always compiled as a library call.

- `BOOL FUNCTION REAL64GT (VAL REAL64 X, Y)`
  Always compiled as a library call.

- `INT FUNCTION IEEECOMPARE (VAL REAL32 X, Y)`
  Always compiled as a library call.

- `INT FUNCTION DIEEECOMPARE (VAL REAL64 X, Y)`
  Always compiled as a library call.

5.4 Transputer-specific predefined routines

5.4.1 General purpose routines

- `PROC CAUSEERROR ()`
  Always compiled inline.

- `PROC RESCHEDULE ()`
  Always compiled inline.

- `PROC LOAD.BYTE.VECTOR (INT here, VAL []BYTE bytes)`

Always compiled inline.

- `PROC LOAD.INPUT.CHANNEL (INT here, CHAN OF ANY in)`
  Always compiled inline.

- `PROC LOAD.INPUT.CHANNEL.VECTOR (INT here, []CHAN OF ANY in)`
  Always compiled inline.

- `PROC LOAD.OUTPUT.CHANNEL (INT here, CHAN OF ANY out)`
  Always compiled inline.

- `PROC LOAD.OUTPUT.CHANNEL.VECTOR (INT here, []CHAN OF ANY out)`
  Always compiled inline.

5.4.2 Block Move routines

-
  ```
  PROC MOVE2D (VAL [][]BYTE source, VAL INT sx, sy,
                [][]BYTE dest, VAL INT dx, dy,
               VAL INT width, length)
  ```

  Compiled inline on a processor which has the *move2d* instructions, otherwise compiled as a library call.

-
  ```
  PROC CLIP2D (VAL [][]BYTE source, VAL INT sx, sy,
                [][]BYTE dest, VAL INT dx, dy,
               VAL INT width, length)
  ```

  Compiled inline on a processor which has the *clip2d* instructions, otherwise compiled as a library call.

-
  ```
  PROC DRAW2D (VAL [][]BYTE source, VAL INT sx, sy,
                [][]BYTE dest, VAL INT dx, dy,
               VAL INT width, length)
  ```

  Compiled inline on a processor which has the *draw2d* instructions, otherwise compiled as a library call.

5.4.3 Cyclic redundancy checking

- `INT FUNCTION CRCWORD (VAL INT data, CRCIn, generator)`
  Compiled inline on a processor which has the *crcword* instruction, otherwise compiled as a library call.

- `INT FUNCTION CRCBYTE (VAL INT data, CRCIn, generator)`
  Compiled inline on a processor which has the *crcbyte* instruction, otherwise compiled as a library call.

5.4.4 Bit manipulation routines

- `INT FUNCTION BITCOUNT (VAL INT Word, CountIn)`
  Compiled inline on a processor which has the *bitcnt* instruction, otherwise compiled as a library call.

- `INT FUNCTION BITREVWORD (VAL INT X)`

Compiled inline on a processor which has the *bitrevword* instruction, otherwise compiled as a library call.

- **`INT FUNCTION BITREVNBITS (VAL INT X, n)`**
  Compiled inline on a processor which has the *bitrevnbits* instruction, otherwise compiled as a library call.

5.4.5 Floating point support routines

- **`INT FUNCTION FRACMUL (VAL INT x, y)`**
  Compiled inline on a processor which has the *fmul* instruction, ie. all 32-bit processors, otherwise compiled as a library call.

- **`INT, INT, INT FUNCTION UNPACKSN (VAL INT X)`**
  Compiled inline on a processor which has the *unpacksn* instruction, otherwise compiled as a library call. Invalid on a 16-bit processor, since the mantissa, etc, of a **`REAL32`** cannot fit into an **`INT`**.

- **`INT FUNCTION ROUNDSN (VAL INT Yexp, Yfrac, Yguard)`**
  Compiled inline on a processor which has the *roundsn* instruction, otherwise compiled as a library call. Invalid on a 16-bit processor, since the mantissa, etc, of a **`REAL32`** cannot fit into an **`INT`**.

5.4.6 Dynamic code loading

-
  ```
  PROC KERNEL.RUN (VAL []BYTE code, VAL INT entry.offset,
                    []INT workspace,
                   VAL INT number.of.parameters)
  ```

  Always compiled inline. Note that the compiler requires that **`number.of.parameters`** is compile-time constant, and has value $\geq$ 3.

6 Summary

The following table summarizes which routines are needed on each processor type. A tick indicates that the routine is required compiled for that processor. The name of another processor (or class) also indicates that the routine is required, but that it can use the one compiled for the indicated processor type, since the code generated for that library routine would be identical. Note that technically this information is an *implementation* issue, not a specification (ie. it depends upon how **`REAL32OP`** etc. are implemented), but it is included here for completeness. To read this as a *specification*, consider any column containing a processor type simply as a tick.

Routine	**T212**	**T225**	**TA**	**TB**	**T414**	**T425**	**T800**
INT16ADD%CHK			✓	TA	TA	TA	✓
INT16SUB%CHK			✓	TA	TA	TA	✓
INT16MUL%CHK			✓	TA	TA	TA	✓
INT16DIV%CHK			✓	TA	TA	TA	✓
INT16REM%CHK			✓	TA	TA	TA	✓
INT16PLUS%CHK			✓	TA	TA	TA	✓
INT16MINUS%CHK			✓	TA	TA	TA	✓
INT16TIMES%CHK			✓	TA	TA	TA	✓
INT16BITAND%CHK			✓	TA	TA	TA	✓
INT16BITOR%CHK			✓	TA	TA	TA	✓
INT16XOR%CHK			✓	TA	TA	TA	✓
INT16LSHIFT%CHK			✓	TA	TA	TA	✓
INT16RSHIFT%CHK			✓	TA	TA	TA	✓
INT16BITNOT%CHK			✓	TA	TA	TA	✓
INT16GT%CHK			✓	TA	TA	TA	✓
INT16EQ%CHK			✓	TA	TA	TA	✓
INT32MUL%CHK	✓	T212					
INT32DIV%CHK	✓	T212					
INT32REM%CHK	✓	T212					
INT64ADD%CHK	✓	T212					
INT64SUB%CHK	✓	T212					
INT64MUL%CHK	✓	T212	✓	TA	TA	TA	✓
INT64DIV%CHK	✓	T212	✓	TA	TA	TA	✓
INT64REM%CHK	✓	T212	✓	TA	TA	TA	✓
INT64PLUS%CHK	✓	T212					
INT64MINUS%CHK	✓	T212					
INT64TIMES%CHK	✓	T212					
INT64BITAND%CHK	✓	T212					
INT64BITOR%CHK	✓	T212					
INT64XOR%CHK	✓	T212					
INT64LSHIFT%CHK	✓	T212					
INT64RSHIFT%CHK	✓	T212					
INT64BITNOT%CHK	✓	T212					
INT64GT%CHK	✓	T212					
INT64EQ%CHK	✓	T212					

Routine	T212	T225	TA	TB	T414	T425	T800
REAL32OPERR%CHK	√	T212	√	√	TB	TB	
REAL32REMERR%CHK	√	T212	√	TA	TA	TA	
REAL32GTERR%CHK	√	T212	√	√	TB	TB	
REAL32EQERR%CHK	√	T212	√	√	TB	TB	
REAL64OPERR%CHK	√	T212	√	TA	TA	TA	
REAL64REMERR%CHK	√	T212	√	TA	TA	TA	
REAL64GTERR%CHK	√	T212	√	TA	TA	TA	
REAL64EQERR%CHK	√	T212	√	TA	TA	TA	

Routine	T212	T225	TA	TB	T414	T425	T800
INT16TOINT64%CHK	√	T212					
INT16TOREAL32%CHK	√	T212					
INT16TOREAL64%CHK	√	T212					
INT32TOINT64%CHK	√	T212					
INT32TOREAL32%CHK	√	T212	√	TA	TA	TA	
INT32TOREAL64%CHK	√	T212	√	TA	TA	TA	
INT64TOINT16%CHK	√	T212					
INT64TOINT32%CHK	√	T212					
INT64TOREAL32%CHK	√	T212	√	TA	TA	TA	
INT64TOREAL64%CHK	√	T212	√	TA	TA	TA	

Routine	T212	T225	TA	TB	T414	T425	T800
REAL32TOINT16%CHK	√	T212					
REAL32TOINT32%CHK	√	T212	√	TA	TA	TA	
REAL32TOINT64%CHK	√	T212	√	TA	TA	TA	
REAL32TOREAL64%CHK	√	T212	√	TA	TA	TA	
REAL64TOINT16%CHK	√	T212					
REAL64TOINT32%CHK	√	T212	√	TA	TA	TA	
REAL64TOINT64%CHK	√	T212	√	TA	TA	TA	
REAL64TOREAL32%CHK	√	T212	√	TA	TA	TA	

Routine	T212	T225	TA	TB	T414	T425	T800
NOTFINITE%O	√	T212					
MINUSX%O	√	T212					
ABS%O	√	T212	√	√	TB	TB	
ISNAN%O	√	T212	√	TA	TA	TA	
ORDERED%O	√	T212	√	TA	TA	TA	
MULBY2%O	√	T212	√	√	TB	TB	
DIVBY2%O	√	T212	√	√	TB	TB	
SQRT%O	√	T212	√	√	TB	TB	
FPINT%O	√	T212	√	√	TB	TB	
SCALEB%O	√	T212	√	√	TB	TB	√
COPYSIGN%O	√	T212	√	TA	TA	TA	√
NEXTAFTER%O	√	T212	√	TA	TA	TA	√
LOGB%O	√	T212	√	TA	TA	TA	√
FLOATING.UNPACK%O	√	T212	√	TA	TA	TA	√
ARGUMENT.REDUCE%O	√	T212	√	√	TB	TB	√

Routine	T212	T225	TA	TB	T414	T425	T800
DNOTFINITE%O	√	T212	√	TA	TA	TA	
DMINUSX%O	√	T212	√	TA	TA	TA	
DABS%O	√	T212	√	TA	TA	TA	
DISNAN%O	√	T212	√	TA	TA	TA	
DORDERED%O	√	T212	√	TA	TA	TA	
DMULBY2%O	√	T212	√	TA	TA	TA	
DDIVBY2%O	√	T212	√	TA	TA	TA	
DSQRT%O	√	T212	√	TA	TA	TA	
DFPINT%O	√	T212	√	TA	TA	TA	
DSCALEB%O	√	T212	√	TA	TA	TA	√
DCOPYSIGN%O	√	T212	√	TA	TA	TA	√
DNEXTAFTER%O	√	T212	√	TA	TA	TA	√
DLOGB%O	√	T212	√	TA	TA	TA	√
DFLOATING.UNPACK%O	√	T212	√	TA	TA	TA	√
DARGUMENT.REDUCE%O	√	T212	√	TA	TA	TA	√

Routine	T212	T225	TA	TB	T414	T425	T800
REAL32OP%O	√	T212	√	√	TB	TB	√
REAL64OP%O	√	T212	√	TA	TA	TA	√
IEEE32OP%O	√	T212	√	TA	TA	TA	√
IEEE64OP%O	√	T212	√	TA	TA	TA	√
REAL32REM%O	√	T212	√	TA	TA	TA	√
REAL64REM%O	√	T212	√	TA	TA	TA	√
IEEE32REM%O	√	T212	√	TA	TA	TA	√
IEEE64REM%O	√	T212	√	TA	TA	TA	√
REAL32EQ%O	√	T212	√	TA	TA	TA	√
REAL64EQ%O	√	T212	√	TA	TA	TA	√
REAL32GT%O	√	T212	√	TA	TA	TA	√
REAL64GT%O	√	T212	√	TA	TA	TA	√
IEEECOMPARE%O	√	T212	√	TA	TA	TA	√
DIEEECOMPARE%O	√	T212	√	TA	TA	TA	√

Routine	T212	T225	TA	TB	T414	T425	T800
MOVE2D%O	√	T212	√	TA	TA		
CLIP2D%O	√	T212	√	TA	TA		
DRAW2D%O	√	T212	√	TA	TA		
BITCOUNT%O	√		√	TA	TA		
CRCWORD%O	√		√	TA	TA		
CRCBYTE%O	√		√	TA	TA		
BITREVWORD%O	√		√	TA	TA		
BITREVNBITS%O	√		√	TA	TA		
FRACMUL%O	√	T212					
UNPACKSN%O	-	-	√				√
ROUNDSN%O	-	-	√				√

7 Interactive debugger support

If the compiler's interactive debugging option is selected (see SW-0062 ("occam 2 Compiler specification")) then the compiler also requires a library containing the debugging input and output routines. The compiler assumes that this library is called `virtual.lib`.

This library must contain the following routines for all processor types:

- `PROC VIRTUAL.OUT.WORD% (VAL INT message.word,VAL INT chan.address)`
  This routine is called to output a word on a channel.

- `PROC VIRTUAL.OUT.BYTE% (VAL BYTE message.byte,VAL INT chan.address)`
  This routine is called to output a byte on a channel.

-
  ```
  PROC VIRTUAL.OUT% (VAL INT message.length,
                     VAL INT chan.address,
                     VAL INT message.address)
  ```

  This routine is called to output an object not of byte or word length, on a channel.

-
  ```
  PROC VIRTUAL.IN% (VAL INT message.length,
                    VAL INT chan.address,
                    VAL INT message.address)
  ```

  This routine is called to perform channel input.

Routine	**T212**	**T225**	**TA**	**TB**	**T414**	**T425**	**T800**
VIRTUAL.OUT.WORD%	$\surd$	T212	$\surd$	TA	TA	TA	$\surd$
VIRTUAL.OUT.BYTE%	$\surd$	T212	$\surd$	TA	TA	TA	$\surd$
VIRTUAL.OUT%	$\surd$	T212	$\surd$	TA	TA	TA	$\surd$
VIRTUAL.IN%	$\surd$	T212	$\surd$	TA	TA	TA	$\surd$