IV. IBM/370 Machine Support Library

NAME

Conventions - the IBM/370 runtime library

FUNCTION

The modules in this section are called upon by code produced by the IBM/370 C compiler, primarily to perform operations that do not exist as instructions in the machine. Functions are described in terms of their assembler interface, since they operate outside the conventional C calling protocol.

The notation used is that of the Idris assembler. To read code correctly for IBM-assemblers, make the following substitutions:

```
_ becomes $
~ becomes #
/ becomes *
: becomes " EQU *"
.byte n becomes " DC AL1(n)"
.word n becomes " DC H'n'"
.long n becomes " DC F'n'" if n is a number
.long n becomes " DC A(n)" if n is an expression containing a symbol
.align 1 becomes " DS OH"
.align 2 becomes " DS OF"
.align 3 becomes " DS OD"
[a-z] becomes [A-Z]
```

Unless explicitly stated otherwise, every function obeys the normal C calling convention that the condition code, register 12 and the floating point registers are not preserved across a call.

Some of the functions use the eight bytes reserved on top of the stack for parameter passing or as temporary storage.

NAME

c~ent - save registers on entering a C function

SYNOPSIS

```
l     12,12(15)
br    12
.word 0
.long value
.long c~ent
.long name
```

FUNCTION

c~ent sets up a new stack frame and saves the registers in the save area. It is designed to be called on entry to a C function, at which time:

72(10)	holds the first argument
13	holds the address of the save area
14	holds the return link
15	holds the base address of the called function

On return from c~ent:

11	holds the base address
13	is the new save area pointer
72(13)	holds the first argument
4(13)	holds the old save area pointer
10	points to the first free double aligned place below the automatic area on the stack
9	points to the automatic area, with an offset of 4096 bytes below the start of the automatic area.

Automatic storage has been allocated by decrementing the stack pointer with the value at 8(15), and is addressed at 4095(9) on down. Note that an extra double (TOS, 0(10)) is allocated on the stack, so that the code can use TOS as a scratch cell. A space for TOS is always allocated on the stack.

RETURNS

c~ent alters registers 10, 9, 11 and 13 to make the new stack frame. The other registers are not necessarily the same as before the call to c~ent (i.e., no parameters can be passed in registers).

EXAMPLE

The C function:

```
COUNT idiot()
    {
    COUNT i;

    return (i);
    }
```

can be written:

```
.align 3
_idiot:
l       12,12(15)
br      12
.word 0
.long 16                / space for i and TOS
.long c~ent
.long 0
/
l       12,4092(9)      / return value
/
l       13,4(13)        / fetch save area
lm      14,11,12(13)    / restore registers
br      14              / return
```

SEE ALSO

c~ents

NAME

c~ents - save registers on entering a C function

SYNOPSIS

```
 l    12,12(15)
 br   12
 .word 0
 .long value
 .long c~ents
 .long name
```

FUNCTION

c~ents sets up a new stack frame and saves the registers in the save area, ensuring that the stack will not creep below _stop. It is designed to be called on entry to a C function, at which time:

72(10)	holds the first argument
13	holds the address of the save area
14	holds the return link
15	holds the base address of the called function

On return from c~ents:

11	holds the base address
13	is the new save area pointer
72(13)	holds the first argument
4(13)	holds the old save area pointer
10	points to the first free double aligned place below the automatic area on the stack
9	points to the automatic area, with an offset of 4096 bytes below the start of the automatic area.

Automatic storage has been allocated by decrementing the stack pointer with the value at 8(15), and is addressed at 4095(9) on down. Note that an extra double (TOS, 0(10)) is allocated on the stack, so that the code can use TOS as a scratch cell. A space for TOS is always allocated on the stack. If stack overflow would occur, the _memerr condition is raised. This will never happen if _stop is set to zero.

RETURNS

c~ents alters registers 10, 9, 11 and 13 to make the new stack frame. The other registers are not necessarily the same as before the call to c~ents (i.e., no parameters can be passed in registers).

EXAMPLE

The C function:

```
COUNT idiot()
    {
    COUNT i;

    return (i);
    }
```

can be written:

```
.align 3
_idiot:
l      12,12(15)
br     12
.word 0
.long 16                / space for i and TOS
.long c~ents
.long 0
/
l      12,4092(9)       / return value
/
l      13,4(13)         / fetch save area
lm     14,11,12(13)     / restore registers
br     14               / return
```

SEE ALSO

c~ent

NAME

c~dtl - convert double to long

SYNOPSIS

```
    std   freg,0(10)
    l     15,a_c~dtl
    balr  14,15
       .
       .
       .
    a_c~dtl:
    .long c~dtl
```

FUNCTION

c~dtl is the internal routine called instead of the missing conversion instruction to convert a double in freg into a long integer in register 12. It does so by separating the fraction from the characteristic, then shifting the fraction until the binary point is at a known fixed place. The long integer immediately to the left of the binary point is delivered, with the same sign as the original double. Truncation occurs towards zero.

RETURNS

c~dtl returns a long in register 12, which is the low-order 32 bits of the integer representation of the double at the top of stack, truncated towards zero. All registers are preserved.

SEE ALSO

c~ltd

NAME
    c~ltd - convert long to double

SYNOPSIS

```
     st    reg,4(10)
     l     15,a_c~ltd
     balr  14,15
        .
        .
        .
     a_c~ltd:
     .long c~ltd
```

FUNCTION
    c~ltd is the internal routine that is called instead of the missing conversion instruction to convert the long in reg to a double in F0. It does so by extending the long to an unpacked double fraction, then normalizing it with a suitable characteristic.

RETURNS
    c~ltd returns the double representation of the long integer value passed as an argument. All registers are preserved.

SEE ALSO
    c~dtl, c~ultd

NAME

c~switch - perform a C switch statement

SYNOPSIS

```
l(r)  12,val
l     14,a_swtab
l     15,a_c~swi
br    15
   .
   .
   .
a_swtab:
.long swtab
a_c~swi:
.long c~switch
```

FUNCTION

c~switch is the code that branches to the appropriate case in a switch statement. It compares val against each entry in swtab until it finds an entry with a matching case value or until it encounters a default entry. swtab consists of zero or more (lbl, value) pairs, where lbl is the address to jump to and value is the case value that is compared against val.

A default entry is signalled by the pair (0, deflbl), where deflbl is the address to jump to if none of the case values match. The compiler always provides a default entry, which points to the statement following the switch if there is no explicit default statement within the switch.

RETURNS

c~switch exits to the appropriate case or default; it never returns.

NAME
    c~uldiv - unsigned long divide

SYNOPSIS

```
     st     left,0(10)
     st     right,4(10)
     l      15,a_c~uld
     balr   14,15
        .
        .
        .
     a_c~uld:
     .long c~uldiv
```

FUNCTION
    c~uldiv divides the unsigned long left by the unsigned long right, producing an unsigned long quotient.

    No check is made for a zero divisor.

RETURNS
    The value returned is the long quotient left / right in register 12. All other registers are preserved.

SEE ALSO
    c~ulmod

NAME
    c~ulmod - unsigned long remainder

SYNOPSIS

```
     st    left,0(10)
     st    right,4(10)
     l     15,a_c~ulm
     balr  14,15
        .
        .
        .
     a_c~ulm:
     .long c~ulmod
```

FUNCTION
    c~ulmod divides the unsigned long left by the unsigned long right, producing an unsigned long remainder.

    No check is made for zero divisor.

RETURNS
    The value returned is the long remainder left / right in register 12. All other registers are preserved.

SEE ALSO
    c~uldiv

NAME

c~ultd - convert unsigned long to double

SYNOPSIS

```
st    reg,4(10)
l     15,a_c~ultd
balr  14,15
    .
    .
    .
a_c~ultd:
.long c~ultd
```

FUNCTION

c~ultd is the internal routine called instead of the missing conversion instruction to convert the unsigned long in reg into a double in floating point register 0. It does so by extending the unsigned long to an unpacked double fraction, then normalizing it with a suitable characteristic.

RETURNS

c~ultd returns the double representation of the unsigned long value passed as an argument. All registers are preserved.

SEE ALSO

c~dtl, c~ltd