IV. Machine Support Library for MC68000

NAME

Conventions - the MC68000 Machine Support Library

FUNCTION

The modules in this section are called upon by code produced by the MC68000 C compiler, primarily to perform operations too lengthy to be expanded in-line. Functions are described in terms of their assembler interface, since they operate outside the conventional C calling protocol.

Unless explicitly stated otherwise, every function does obey the normal C calling convention that the condition code and registers d0, d1, d2, d6, d7, a0, a1, and a2 are not preserved across a call.

NAME

check - check stack on entering a C function

SYNOPSIS

```
move.l  $-nautos,d0
jsr     a.check
```

FUNCTION

a.check ensures that d0+sp is not lower than _stop, to check for stack overflow.

If stack overflow would occur, the _memerr condition is raised. This will never happen if _stop is set to zero.

RETURNS

Nothing.

EXAMPLE

The C function:

```
COUNT idiot()
    {
    COUNT i;

    return (i);
    }
```

can be written:

```
idiot:
    move.l  #-24,d0
    jsr     a.check
    link    a6,#-2
    move.w  -2(a6),d7
    ext.l   d7
    unlk    a6
    rts
```

SEE ALSO

_stop

NAME

count - counter for profiler

SYNOPSIS

```
        lea     lbl,a0
        jsr     a.count
```

FUNCTION

a.count is the function called on entry to each function when the compiler is run with the flag "-p" given to p2.68k. lbl is the address of a pointer variable, initially NULL, that is used by a.count to aid in its bookkeeping.

On entry to a.count, the return link is always presumed to be 12 bytes beyond a function entry point.

RETURNS

Nothing.

NAME

dadd - add double into double

SYNOPSIS

```
        lea     right,a0    OR  clr.l   a0
        jsr     a.1add      OR  jsr     a.6add
```

FUNCTION

a.1add and a.6add are the internal routines called to add the double at right into a double accumulator, called left, which is d1/d2 for a.1add or d6/d7 for a.6add; if a0 is NULL, the other accumulator is taken as the right operand. The addition is performed without destroying any volatile registers, so the call can be used much like an ordinary machine instruction.

If right is zero, left is unchanged (x+0); if left is zero, right is copied into it (0+x). Otherwise the number with the smaller characteristic is shifted right until it aligns with the other and the addition is performed algebraically. The answer is rounded.

RETURNS

dadd replaces its left operand with the closest internal representation to the rounded sum of its operands. Volatile registers and the stack are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

ddiv, dmul, dsub, repk, unpk

BUGS

It doesn't check for characteristics differing by huge amounts, to save shifting. If the right operand is zero, an unnormalized left operand is left unchanged.

NAME

dcmp - compare two doubles

SYNOPSIS

```
        lea     right,a0    OR  clr.l   a0
        jsr     a.1cmp      OR  jsr     a.6cmp
```

FUNCTION

a.1cmp and a.6cmp are the internal routines called to compare the double at right with a double accumulator, called left, which is d1/d2 for a.1cmp or d6/d7 for a.6cmp; if a0 is NULL, the other accumulator is taken as the right operand. The comparison is performed without destroying any volatile registers, so the call can be used much like an ordinary machine instruction. The comparison involves no floating arithmetic and so is comparatively fast. -0 compares equal with +0.

RETURNS

dcmp sets the NZ condition codes to reflect the difference (left-right). Volatile registers and the stack are preserved.

SEE ALSO

dsub

BUGS

Unnormalized zeros must have at least the leading four fraction bits zero.

NAME

ddiv - divide double into double

SYNOPSIS

```
        lea     right,a0    OR  clr.l   a0
        jsr     a.1div      OR  jsr     a.6div
```

FUNCTION

a.1div and a.6div are the internal routines called to divide the double at right into a double accumulator, called left, which is d1/d2 for a.1div or d6/d7 for a.6div; if a0 is NULL, the other accumulator is taken as the right operand. The division is performed without destroying any volatile registers, so the call can be used much like an ordinary machine instruction.

If right is zero, left is set to the largest representable floating number, appropriately signed (x/0); if left is zero, it is unchanged (0/x). Otherwise the right fraction is divided into the left and the right exponent is subtracted from that of the left. The sign of a non-zero result is negative if the left and right signs differ, else it is positive. The result is rounded.

RETURNS

ddiv replaces its left operand with the closest internal representation to the rounded quotient (left/right), or a huge number if right is zero. Volatile registers and the stack are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

dadd, dmul, dsub, repk, unpk

BUGS

If the left operand is an unnormalized zero, it is left unchanged.

NAME

dmul - multiply double into double

SYNOPSIS

```
        lea     right,a0    OR  clr.l   a0
        jsr     a.1mul      OR  jsr     a.6mul
```

FUNCTION

a.1mul and a.6mul are the internal routines called to multiply the double at right into a double accumulator, called left, which is d1/d2 for a.1add or d6/d7 for a.6add; if a0 is NULL, the other accumulator is taken as the right operand. The multiplication is performed without destroying any volatile registers, so the call can be used much like an ordinary machine instruction.

If either right or left is zero, the result is zero (0*x, x*0). Otherwise the right fraction is multiplied into the left and the right exponent is added to that of the left. The sign of a non-zero result is negative if the left and right signs differ, else it is positive. The result is rounded.

RETURNS

dmul replaces its left operand with the closest internal representation to the rounded product of its operands. Volatile registers and the stack are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

dadd, ddiv, dsub, repk, unpk

BUGS

If the left operand is an unnormalized zero, it is left unchanged.

NAME

dsub - subtract double from double

SYNOPSIS

```
lea     right,a0    OR  clr.l   a0
jsr     a.1sub      OR  jsr     a.6sub
```

FUNCTION

a.1sub and a.6sub are the internal routines called to subtract the double at right from a double accumulator, called left, which is d1/d2 for a.1sub or d6/d7 for a.6sub; if a0 is NULL, the other accumulator is taken as the right operand. The subtraction is performed without destroying any volatile registers, so the call can be used much like an ordinary machine instruction.

dsub copies its right operand, negates the copy, and enters dadd.

RETURNS

dsub replaces its left operand with the closest internal representation to the rounded difference (left-right). Volatile registers and the stack are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

dadd, dcmp, ddiv, dmul, repk, unpk

BUGS

-0 - 0 and -0 - -0 return -0.

NAME

dtf - convert double to float

SYNOPSIS

```
jsr     a.1dtf      OR  jsr     a.6dtf
```

FUNCTION

a.1dtf and a.6dtf are the internal routines called to convert the double accumulator, which is d1/d2 for a.1dtf or d6/d7 for a.6dtf, to a float in the more significant half of the accumulator, i.e., d1 or d6.

RETURNS

dtf returns a float in the more significant half of the accumulator used. Volatile registers and the stack are preserved.

SEE ALSO

ftd

BUGS

No checks are made for overflow or underflow, which give garbage results.

NAME

dtl - convert double to long

SYNOPSIS

```
jsr     a.1dtl      OR  jsr     a.6dtl
```

FUNCTION

a.1dtl and a.6dtl are the internal routines called to convert the double accumulator, which is d1/d2 for a.1dtl or d6/d7 for a.6dtl, to a long integer in d0. They do so by calling unpk, to separate 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 toward zero.

RETURNS

dtl returns a long in d0, which is the low-order 32 bits of the integer representation of the double in the accumulator used, truncated toward zero. Volatile registers and the stack are preserved.

SEE ALSO

ltd, ultd

NAME

ftd - convert float to double

SYNOPSIS

```
jsr     a.1ftd      OR  jsr     a.6ftd
```

FUNCTION

a.1ftd and a.6ftd are the internal routines called to convert a float in the more significant half of a double accumulator, which is d1/d2 for a.1ftd or d6/d7 for a.6ftd, to a double in the accumulator.

RETURNS

ftd returns a double in the accumulator used. The NZ condition codes are set to reflect the result. Volatile registers and the stack are preserved.

SEE ALSO

dtf

NAME

ldiv - divide long by long

SYNOPSIS

```
move.l  n,-(sp)
move.l  d,-(sp)
jsr     a.ldiv
```

FUNCTION

a.ldiv divides the long n by the long d to obtain the long quotient. The sign of a non-zero result is negative only if the signs of n and d differ. No check is made for division by zero, which currently gives a quotient of -1 or +1.

RETURNS

The value returned is the long quotient of n / d on the stack. Volatile registers are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

lmod, lmul, uldiv, ulmod

NAME

lmod - divide long by long and return remainder

SYNOPSIS

```
        move.l  n,-(sp)
        move.l  d,-(sp)
        jsr     a.lmod
```

FUNCTION

a.lmod divides the long n by the long d to obtain the long remainder. The sign of a non-zero result is negative only if the sign of n is negative. No check is made for division by zero, which currently gives a remainder equal to the value of n.

RETURNS

The value returned is the long remainder of n / d on the stack. Volatile registers are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

ldiv, lmul, uldiv, ulmod

NAME

lmul - multiply long by long

SYNOPSIS

```
move.l  l,-(sp)
move.l  r,-(sp)
jsr     a.lmul
```

FUNCTION

a.lmul multiplies the long l by the long r to obtain the long product. The sign of a non-zero result is negative only if the signs of l and r differ. No check is made for overflow, which currently gives the low order 32 bits of the correct product.

RETURNS

The value returned is the long product l * r on the stack. Volatile registers are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

ldiv, lmod, uldiv, ulmod

NAME

ltd - convert long to double

SYNOPSIS

```
        move.l  n,d0
        jsr     a.1ltd      OR  jsr     a.6ltd
```

FUNCTION

a.1ltd and a.6ltd are the internal routines called to convert the long integer in d0 to a double in an accumulator, which is d1/d2 for a.1ltd or d6/d7 for a.6ltd. They do so by extending the long to an unpacked double fraction, then calling repk with a suitable characteristic. The conversion is performed without destroying any volatile registers, so the call can be used much like an ordinary machine instruction.

RETURNS

ltd writes the double equivalent of the long in d0 into the specified accumulator. Volatile registers and the stack are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

dtl, repk, ultd

NAME

ltf - convert long to float

SYNOPSIS

```
move.l  n,d0
jsr     a.ltf
```

FUNCTION

a.ltf is the internal routine called to convert the long integer in d0 to a float in d0. It does so by calling ltd and dtf in turn, and without destroying any volatile registers, so that the call can be used much like an ordinary machine instruction.

RETURNS

ltf writes the float equivalent of the long in d0 into d0. Volatile registers and the stack are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

dtf, ltd, ultf

NAME

repk - repack a double number

SYNOPSIS

```
move.l  char,-(sp)
pea     frac
jsr     a.repk
addq.l  #8,sp
```

FUNCTION

a.repk is the internal routine called by various floating runtime routines to pack a signed fraction at frac and binary characteristic char into a standard form double representation. The fraction occupies two four-byte integers, starting at frac, and may contain any value; there is an assumed binary point immediately to the right of the most significant byte. The characteristic is 1023 plus the power of two by which the fraction must be multiplied to give the proper value.

If the fraction is zero, the resulting double is all zeros. Otherwise the fraction is forced positive and shifted left or right as needed until the most significant bit is at the 020 position in byte #1 (counting from zero), with the characteristic being incremented or decremented as appropriate. The fraction is then rounded to 53 binary places. If the resultant characteristic can be properly represented in a double, it is put in place and the sign is set to match the original fraction sign. If the characteristic is zero or negative, the double is all zeros. Otherwise the characteristic is too large, so the double is set to the largest representable number, and is given the sign of the original fraction.

RETURNS

repk replaces the two four-byte integers of the fraction with the double representation. The value of the function is VOID, i.e., garbage. The NZ condition codes are set to reflect the result, however, which is nonstandard.

SEE ALSO

unpk

BUGS

Really large magnitude values of char might overflow during normalization and give the wrong approximation to an out of range double value.

NAME

switch - perform C switch statement

SYNOPSIS

```
        move.l  val,d7
        lea     swtab,a0
        jmp     a.switch
```

FUNCTION

a.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 entries consist of zero or more (lbl, value) pairs, where lbl is the (nonzero) address to jump to and value is the case value that must match 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

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

NAME

uldiv - divide unsigned long by unsigned long

SYNOPSIS

```
        move.l  n,-(sp)
        move.l  d,-(sp)
        jsr     a.uldiv
```

FUNCTION

a.uldiv divides the long n by the long d to obtain the unsigned long quotient. No check is made for division by zero, which currently gives a quotient of -1.

RETURNS

The value returned is the long unsigned quotient of n / d on the stack. Volatile registers are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

ldiv, lmod, lmul, ulmod

NAME

ulmod - unsigned long remainder

SYNOPSIS

```
move.l  n,-(sp)
move.l  d,-(sp)
jsr     a.ulmod
```

FUNCTION

a.ulmod divides the long n by the long d to obtain the unsigned long remainder. No check is made for division by zero, which currently gives a remainder of n.

RETURNS

The value returned is the long unsigned remainder of n / d on the stack. Volatile registers are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

ldiv, lmod, lmul, uldiv

NAME

ultd - convert unsigned long to double

SYNOPSIS

```
move.l  n,d0
jsr     a.1ultd     OR  jsr     a.6ultd
```

FUNCTION

a.1ultd and a.6ultd are the internal routines called to convert the unsigned long integer in d0 to a double in an accumulator, which is d1/d2 for a.1ultd or d6/d7 for a.6ultd. They do so by extending the long to an unpacked double fraction, then calling repk with a suitable characteristic. The conversion is performed without destroying any volatile registers, so the call can be used much like an ordinary machine instruction.

RETURNS

ultd writes the double equivalent of the unsigned long in d0 into the specified accumulator. Volatile registers and the stack are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

dtl, ltd, repk

NAME

ultf - convert unsigned long to float

SYNOPSIS

```
move.l  n,d0
jsr     a.ultf
```

FUNCTION

a.ultf is the internal routine called to convert the unsigned long integer in d0 to a float in d0. It does so by calling ultd and dtf in turn, and without destroying any volatile registers, so that the call can be used much like an ordinary machine instruction.

RETURNS

ultf writes the float equivalent of the unsigned long in d0 into d0. Volatile registers and the stack are preserved. The NZ condition codes are set to reflect the result.

SEE ALSO

dtf, ltf, ultd

NAME

unpk - unpack a double number

SYNOPSIS

```
        pea     double
        pea     frac
        jsr     a.unpk
        addq.l  #8,sp
```

FUNCTION

a.unpk is the internal routine called by various floating runtime routines to unpack a double at double into a signed fraction at frac and a characteristic. The fraction consists of two four-byte integers at frac; the binary point is immediately to the right of the most significant byte. If the double at double is not zero, unpk guarantees that the most significant fraction byte is exactly equal to 1, and the least significant fraction byte has three zero bits for use as guard, rounding, and sticky bits.

The characteristic returned is 1023 plus the power of two by which the fraction must be multiplied to give the proper value; it will be zero for any flavor of zero at double, i.e., having a characteristic of zero, irrespective of other bits.

RETURNS

unpk writes the signed fraction as two four-byte integers starting at frac and returns the characteristic in d7 as the value of the function.

SEE ALSO

repk