fpsp

COLLABORATORS			
	TITLE : fpsp		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 9, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 fpsp **1**

1.1 fpsp.doc 1

1.2 fpsp.resource/--Background-- 1

1.3 fpsp.resource/FPSPMonadic 2

1.4 fpsp.resource/FPSPDyadic 3

Chapter 1

fpsp

1.1 fpsp.doc

```
--Background--
FPSPMonadic()
FPSPDyadic()
```

1.2 fpsp.resource/--Background--

```
  PURPOSE

This resource contains the emulation routines for FPU instructions
which are non-native for the 68040 or the 68060 processor.
This resource is - as all resources - not disk based. Instead, it
is build by the 68040 or 68060.library on startup. The CPU driver
libraries install the "FPSP" code as part of their initialization
process, such that every non-impelemented FPU instruction will be
emulated properly. However, this means that for every such
instruction, the CPU has to go thru exception processing, and will
therefore stop multitasking when emulating a non-implemented in-
struction. Calling the fpsp routines directly thru the new
fpsp.resource will prevent this overhead and will gain some speed.

A separate patch - the FastIEEE program - will patch the system
math libraries to make use of this resource.

The fpsp.resource is re-entrant, interrupt-callable code. The
resource offers only two functions which emulate monadic resp.
dyadic functions of the 68881/68882 FPU on a 68040 FPU.

These functions are designed to be stub functions for assembly
language, they are not designed to be callable from C or any
other high-level language. Typical applications would call these
routines indirectly thru a link library which provides all
requires parameters. Direct calling from within application
programs is not desired.

Moreover, the FPSP routines *do not* generate FPU exceptions them-
```

```
selves for speed reasons. If this is desired, the calling routine
has to check the fpcr and the fpsr FPU registers manually and
has to generate the exceptions itself.
```

1.3 fpsp.resource/FPSPMonadic

```
  NAME
FPSPMonadic - emulate a monadic 68881/68882 instruction

  SYNOPSIS
storeflag = FPSPMonadic ( opword, operand );
d0          d0    fp0

BYTE FPSPMonadic ( UWORD , IEEEExtended );

  FUNCTION
This function emulates the 68881/68882 monadic instruction
passed in in register d0, and performs the desired mathematical
operation on FP0. It returns the result in the FPU register FP0
and FP1.

  INPUTS
opword  - The extension word of a valid 68881/68882
    instruction. Valid extension words are found in
    the motorola family guide or the 68881/68882
    manual.
    Note that this is actually the SECOND (!) opword
    of the 68881/68882 instruction.
    The bits of the opword have to be specified as
    follows:

    Bits 15 and 13 must be cleared. The fpsp.resource
    does not emulate FPU control instructions.

    R/M:  Source operand mode,  bit 14.
        Must be set to zero.
    Source: Source specifier,   bits 12 to 10.
        Must be set to zero.
    Dest: Destination specifier,  bits 9 to 7.
        Must be set to zero.
    Inst: Instruction field,  bits 5 to 0.
        Must be filled with the opcode of the
        desired instruction, see the motorola
        documents for the list. For the special
        case of FPSINCOS, the FPC field, bits
        2 to 0, has to be set to 001 to encode
        FP1 as secondary result register.

operand - A valid IEEE extended precision number as argument
    to the function to be performed.

Secondary inputs are delivered in the FPU registers FPCR and FPSR,
namely to select the rounding precision and the behaiviour in case
of invalid arguments or infinite results.
```

```
  RESULTS
storeflag-  This is set to 0 in case the FPSP routine could
    generate a result. THIS DOES NOT MEAN that the
    arguments are valid. In case all exceptions are
    disabled, the storeflag will always be returned as
    zero, but the result might be infinite or a NAN.

Secondary results are delivered in the FPU registers FP0, FP1 and
FPSR.

FP0 contains the (primary) result of the 68881/68882 instruction
in case the storeflag is 0. *IT MIGHT WELL BE A NAN OR AN INF*

In case the storeflag is non-zero, FP0 is undefined and the FPSP
routines indicate that an exception should be generated by the
caller. This happens only if an exception condition was detected
and the corresponding exception has been enabled in the FPCR
passed in. In all other cases a NAN or a INF will be returned
and the storeflag will indicate a valid result.

FP1 contains a secondary result, if any. The only case were FP1
is used is the FSINCOS instruction where the sine is returned in
FP0 and the cosine in FP1. Otherwise undefined.

FPSR contains the FPU status similar to what a real 68881/68882
would have returned.

  NOTES
This function should not be called directly. It should either be
called by the system math libraries, or by compiler link libaries.

Note that a non-zero store flag indicates that an exception
condition was detected. NOTE THAT THE FPSP CODE DOES NOT CAUSE
AN EXCEPTION ITSELF. It is the responsibility of the caller to
check the FPSR, the FPCR and the storeflag to generate this
exception itself, if desireable. In case the storeflag is non-
zero, no useful result is returned in FP0 or FP1.

Useful instruction opwords can be found in the motorola
documentation, either the MC68K family guide, or in the 68881/
68882 manual.

This function does not require a6 to be loaded with the FPSP
resource base. It is interrupt-callable.

  BUGS

  SEE ALSO
libraries/68040.library, libraries/68060.library,
FPSPDyadic,
the Motorola 68881/68882 manual, the MC68K family guide.
```

1.4 fpsp.resource/FPSPDyadic

```
  NAME
FPSPDyadic  - emulate a dyadic 68881/68882 instruction

  SYNOPSIS
storeflag = FPSPDyadic ( opword, operand1, operand2 );
d0         d0    fp0    fp1

BYTE FPSPDyadic ( UWORD , IEEEExtended , IEEEExtended );

  FUNCTION
This function emulates the 68881/68882 dyadic instruction
passed in in register d0, and performs the desired mathematical
operation on FP0 and FP1. It returns the result in the FPU
register FP0.

  INPUTS
opword  - The extension word of a valid 68881/68882
    instruction. Valid extension words are found in
    the motorola family guide or the 68881/68882
    manual.
    Note that this is actually the SECOND (!) opword
    of the 68881/68882 instruction.
    The bits of the opword have to be specified as
    follows:

    Bits 15 and 13 must be cleared. The fpsp.resource
    does not emulate FPU control instructions.

    R/M:  Source operand mode,  bit 14.
        Must be set to zero.
    Source: Source specifier, bits 12 to 10.
        Must be set to one.
    Dest: Destination specifier,  bits 9 to 7.
        Must be set to zero.
    Inst: Instruction field,  bits 5 to 0.
        Must be filled with the opcode of the
        desired instruction, see the motorola
        documents for the list. For the special
        case of FPSINCOS, the FPC field, bits
        2 to 0, has to be set to 001 to encode
        FP1 as secondary result register.

operand1- A valid IEEE extended precision number as argument
    to the function to be performed. This is the source
    operand of dyadic instructions.

operand2- The destination operand of the dyadic 68881/68882
    instructions.

Secondary inputs are delivered in the FPU registers FPCR and FPSR,
namely to select the rounding precision and the behaiviour in case
of invalid arguments or infinite results.

  RESULTS
storeflag-  This is set to 0 in case the FPSP routine could
    generate a result. THIS DOES NOT MEAN that the
```

arguments are valid. In case all exceptions are disabled, the storeflag will always be returned as zero, but the result might be infinite or a NAN.

Secondary results are delivered in the FPU registers FP0, FP1 and FPSR.

FP0 contains the (primary) result of the 68881/68882 instruction in case the storeflag is 0. *IT MIGHT WELL BE A NAN OR AN INF*

In case the storeflag is non-zero, FP0 is undefined and the FPSP routines indicate that an exception should be generated by the caller. This happens only if an exception condition was detected and the corresponding exception has been enabled in the FPCR passed in. In all other cases a NAN or a INF will be returned and the storeflag will indicate a valid result.

FP1 contains a secondary result, if any. As there is currently no dyadic opcode with a secondary result, this register should be ignored.

FPSR contains the FPU status similar to what a real 68881/68882 would have returned.

NOTES

This function should not be called directly. It should either be called by the system math libraries, or by compiler link libaries.

Note that a non-zero store flag indicates that an exception condition was detected. NOTE THAT THE FPSP CODE DOES NOT CAUSE AN EXCEPTION ITSELF. It is the responsibility of the caller to check the FPSR, the FPCR and the storeflag to generate this exception itself, if desireable. In case the storeflag is non-zero, no useful result is returned in FP0 or FP1.

Useful instruction opwords can be found in the motorola documentation, either the MC68K family guide, or in the 68881/68882 manual.

This function does not require a6 to be loaded with the FPSP resource base. It is interrupt-callable.

BUGS

Note that the encoding of the instruction to be performed indicates that the source is delivered in register fp1, even though this function takes the source argument in register fp0 and delivers the destination in fp0.

SEE ALSO

libraries/68040.library, libraries/68060.library, FPSPMonadic, the Motorola 68881/68882 manual, the MC68K family guide.