rf68000

(c) 2022 Robert Finch

# Overview

The rf68000 processing core is similar to a 68010 cpu and may execute software written for the 68000. The core is not bus cycle or cycle by cycle compatible to the 68000. Programs that depend on the 68000’s prefetch may not execute correctly. There are also issues with the BCD instructions and DIVS not operating identical to a 68000.

# Programming Model

The rf68000 has the 68000’s programming model. Eight data registers, and eight address registers. Two stack pointers, program counter and status register.

### Definition of Core Control and Status Registers (CSR’s)

The following control registers are available via the MOVEC instruction:

|  |  |  |  |
| --- | --- | --- | --- |
| Regno | Register | R/W |  |
| $000 | SFC | RW | Source function code |
| $001 | DFC | RW | Destination function code |
| $003 | ASID | RW | Address space identifier |
| $010 | APC | RW | Application control register (currently not used) |
| $011 | CPL | RW | Current privilege level (currently not used) |
| $012 | TR | RW | Current Thread register |
| $013 | TCBA | RW | Pointer to thread control block array |
| $014 | MMUS | RW | MMU circuit select address |
| $015 | IOS | RW | IO circuit select address |
| $016 | IOPS | RW | IO permissions circuit select address |
| $800 | USP | RW | User stack pointer |
| $801 | VBR | RW | Vector base register – must be long word aligned |
| $FE0 | CORENO | R | sometimes called a hartid, identifies the core in a multi-core system |
| $FF0 | TICK | R | tick count, counts up continuously after external reset. |
| $FF8 | ICNT | R | instruction count |
|  |  |  |  |

### TR – Thread Register

This register is a reference register that contains the thread number of the currently running thread.

|  |  |
| --- | --- |
| 31 12 | 11 0 |
| ~ | Thread Number12 |

### TCBA – Thread Control Block Array Pointer

This register contains a pointer to the thread control block (TCB) array.

|  |  |
| --- | --- |
| 31 8 | 7 0 |
| TCB pointer31..8 | 08 |

### ASID – Address Space Identifier

This register contains the address space identifier. The address space identifier is used by the MMU and IO permissions bitmap to provide a range of addresses for a process.

|  |  |
| --- | --- |
| 31 8 | 7 0 |
| ~31..8 | ASID8 |

### IOS – IO Select Register

The location of IO is determined by the contents of the IOS control register. The select is for a 1MB region. This address is a virtual address. The low order 20 bits of this register should be zero and are ignored. The corresponding physical address programmed into the MMU should be one of $FDx00000, $Axx00000 or $Bxx00000 as that is what the network controller forwards.

|  |  |
| --- | --- |
| 31 20 | 19 0 |
| Physical Address31..20 | 020 |

# Hardware

## Bus

The RF68000 uses 32-bit data and address busses. With a wider data bus some of the instructions accessing 32-bit data execute more quickly than they would for the 68k. The RF68000 uses a WISHBONE bus rather than the asynchronous 68k bus. WISHBONE has a minimum bus cycle time of two clock cycles versus the 68k’s four.

## Endian

The RF68000 has a configuration option to operate as a little-endian machine. The least significant bytes are stored at the lowest address. This differs from the 68k which a big-endian machine. This impacts the order of data stored in the system and specification of immediate constants. There is currently no software available for little-endian mode.

# Small Memory Management Unit

## Overview

There is a single small MMU in the system which performs address mapping for all cores. The small MMU supports up to sixty-four address spaces each with 512 pages. The mapping table is 128kB in size. The pages it maps are 64kB in size. The address space may have a maximum of 32MB of memory mapped. The high order bits of the virtual address provided by a processing core are mapped into a different address range using a lookup table. The lookup table is in a dedicated memory. Layout of the table is as follows:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **31 18** | **17** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 000 | ~ | W | X | A31 | A30 | A29 | A28 | A27 | A26 | A25 | A24 | A23 | A22 | A21 | A20 | A19 | A18 | A17 | A16 |
| … |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1FF | ~ | W | X | A31 | A30 | A29 | A28 | A27 | A26 | A25 | A24 | A23 | A22 | A21 | A20 | A19 | A18 | A17 | A16 |

W: indicates memory is writable

X: indicates memory is executable

## MMUS – MMU Select Register

The location of the SMMU mapping table for an address space is determined from the contents of the MMU select control register. This address is a physical address for which the low order 20 bits should be zero and are ignored. The address should be one of $FDx00000, $Axx00000 or $Bxx00000 as that is what the network controller forwards.

|  |  |
| --- | --- |
| 31 20 | 19 0 |
| Physical Address31..20 | 020 |

# IO Permission Bitmap

## Overview

The IO permission bitmap is a dedicated special purpose RAM located somewhere in the memory system. The IO permission bitmap is a bitmap of IO port locations visible in an address space. All IO is located within a 1MB region of the physical address space. The address of I/O is determined from a block translation register the IOS. This region is divided into 4096 256-byte pages which is the granularity of the bitmap. There is a bit in the bitmap for each page. If the bit is set then the address space has access to the IO ports on that page; otherwise, the IO is inaccessible. The system will generate a bus error if an access to an inaccessible IO port is attempted.

The bitmap is represented as an array of 32-bit words, 128 words per address space. The least significant bit of the first word controls access to the first IO page at IOS+$000. The second bit controls access to the second IO page at IOS+$100, and so on. The 32-bit words are group into 128-word groups for each address space. The first group of 128 words is for address space zero, the next group for address space one, and so on. There are a total of 8192 words or 32kB used for the IO permission bitmap.

The location of the IO permission bitmap is determined by the contents of the IOPS control register.

## IOPS – IOP Select Register

The location of the IO permission bitmap is determined by the contents of the IOPS control register. The table itself is 32kB bytes in size. This address is a virtual address. The low order 16 bits of this register should be zero and are ignored. The corresponding physical address programmed into the MMU should be one of $FDxx0000, $Axxx0000 or $Bxxx0000 as that is what the network controller forwards.

|  |  |
| --- | --- |
| 31 16 | 15 0 |
| Physical Address31..16 | 016 |

# Altered Instructions

## BTST, BCHG, BCLR, BSET

**Description:**

If enabled via a configuration definition, BTST, BCHG, BCLR, and BSET are all capable of working with bit-pairs in addition to individual bits.

Since the bit manipulated is specified modulo the instruction size, there are extra “unused” bits in the bit number specifier. For bit-pair operations bit #7 of the bit number specifier indicates that the instruction is a bit-pair operating instruction, in which case the bits normally specifying a particular bit to test or manipulate, then specify a bit-pair to operate on.

**Supported Sizes:** .B .L

**Bit-Pair Flag Updates:**

Bit pair instructions update the z, c, n and v flags with the status of the previous value of the bit-pair. This makes it possible to branch on any combination of bits.

Zf: set if bit pair is 00 otherwise cleared

Cf: set if bit pair is 01 otherwise cleared

Nf: set if bit pair is 10 otherwise cleared

Vf: set if bit pair is 11 otherwise cleared

# Additional Instructions Beyond the 68000

## BCD2BIN

**Description:**

Converts an eight-digit packed BCD number to binary. The value in the data register is treated as a packed BCD number. The conversion uses adding and shifting to convert the number. Each digit is multiplied by the corresponding power of ten. Illegal BCD digits will generate unspecified results.

**Supported Sizes:** .L

**Operation:** Dn = Binary(Dn)

**Flag updates:**

Zero: Set if the result is zero, cleared otherwise

**Instruction Format:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 12 | 11 8 | 7 6 | 5 3 | 2 0 |  | 15 12 | 11 8 | 7 4 | 3 0 |
| Ah4 | 24 | 32 | 04 | Dn3 |  | 04 | 04 | 04 | 14 |

## BIN2BCD

**Description:**

Converts a binary number to packed BCD. The value in the data register is treated as an unsigned integer. The double-dabble algorithm is used. 20 or more clock cycles may be required to perform the conversion.

**Supported Sizes:** .L

**Operation:** Dn = BCD(Dn)

**Flag updates:**

Overflow: Set if the resulting packed BCD number will not fit into 32-bits (8 digits), otherwise cleared.

Zero: Set if the result is zero, cleared otherwise

**Instruction Format:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 12 | 11 8 | 7 6 | 5 3 | 2 0 |  | 15 12 | 11 8 | 7 4 | 3 0 |
| Ah4 | 24 | 32 | 04 | Dn3 |  | 04 | 04 | 04 | 04 |

## MOVEC

**Description:**

This instruction operates in a manner compatible with the 68010’s MOVEC instruction. See the programming model section for a table of supported CSRs.

# Floating-Point Operations

## Overview

The core repurposes the packed BCD floating-point instructions to implement triple precision densely-packed-decimal floating point. The core may be built without floating point to reduce the size of the core, by commenting out the SUPPORT\_DECFLT define.

To support the cores floating-point, the decimal floating-point primitives found in the ft816float project at opencores.org are used.

The following decimal floating-point functions are supported in the core:

FADD, FSUB, FMUL, FDIV, FNEG, FSCALE, FCMP, FTST, FBcc, FMOVE

The instructions may not be completely implemented.

## Representation

The core uses a 96-bit densely packed decimal floating-point representation.

*96-bit values are more compact than 128-bit ones which reduces the amount of data being transferred. They have enough significant digits for a wide variety of applications. 64-bit values are not sufficient for some applications. The question then is how much larger of a representation to use. 80-bits is popular, offering about 19 significant digits which is good for a wide variety of applications. The author wanted to break the 20 digit barrier though.*

|  |  |  |  |
| --- | --- | --- | --- |
| 95 | 94 90 | 89 80 | 79 0 |
| S | Combo5 | Exponent10 | Significand80 |

The significand stores 25 densely packed decimal digits. One whole digit before the decimal point.

The exponent is a power of ten as a binary number with an offset of 1535. Range is 10-1535 to 101536

Please review the IEEE docs for the representation.

## Floating-Point Status Register

rf68000 makes use of the high order four bits of the status register to hold low order quotient bits seven to nine. The low order quotient bits are a 10-bit densely-packed-decimal value containing 3 BCD digits.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 28 | 27 24 | 23 16 | 15 8 | 7 0 |
| Quotient9..7 | NZIA | Sign,Quotient6..0 | Exception | Accrued Exception |