F-CPU Project **F-CPU**

Architecture Guide

by the F-CPU Design Team.

Draft 0.19990904

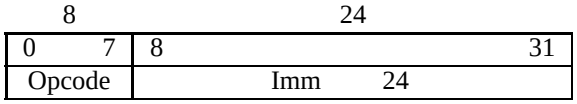
Contents

1 Foreword

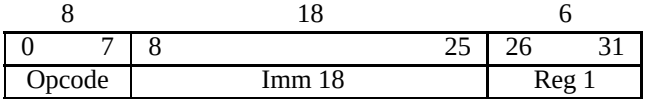
This is preliminary... Anything can be changed without notice.

2 Instruction formats

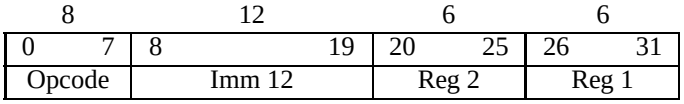
2.1 Format 0



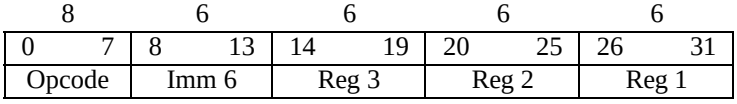
2.2 Format 1



2.3 Format 2



2.4 Format 3



2.5 Size flags

In some opcodes the flags can contain a 'size' parameter that define the size of the operand on which the operation should take place. See the following table:

Flags	Size (byte)	Suffix	Name
00	1	B	Byte
01	2	D	Double-Byte
10	4	Q	Quad-Byte
11	8	(none)	Octa-Byte (Word)

3 Arithmetic

3.1 Core Arithmetic

3.1.1 add

Addition/Substraction

add %a , %b, %c

Computes $\%c = \%a + \%b$.

0 7	8 13	14 19	20 25	26 31
OP_ADD	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[u]	Defines if the operands are unsigned
11	[s]	Defines if the operation is SIMD (*)
12	[n]	Defines if the the second operand is negated.
13		Unused.

sub %a, %b, %c

Computes $\%c = \%a - \%b$.
sub is just an alias for **add.n** .

3.1.2 mul

Multiplication

mul %a, %b, %c

Computes $\%c = \%a \times \%b$.

0 7	8 13	14 19	20 25	26 31
OP_MUL	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[u]	Defines if the operands are unsigned
11	[s]	Defines if the operation is SIMD (*)

3.1.3 div

Division

div %a, %b, %c

Computes $\%c = \%a / \%b$.

0 7	8 13	14 19	20 25	26 31
OP_DIV	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[u]	Defines if the operands are unsigned
11	[s]	Defines if the operation is SIMD (*)

3.2 Optional Arithmetic

3.2.1 addi and subi

Addition/Substraction Immediate

addi %a, i8, %b

Compute $\%b = \%a + i8$.

0 7	8 13	14 19	20 25	26 31
OP_ADDI	FLAGS	imm8	%a	%b

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[u]	Defines if the operands are unsigned
11	[s]	Defines if the operation is SIMD (*)

subi %a, i8, %b

Computes $\%b = \%a - i8$
sub is just an alias for **addi.n** .

3.2.2 muli

Multiplication Immediate

muli %a, i8, %b

Computes $\%b = \%a \times i8$.

0 7	8 13	14 19	20 25	26 31
OP_MULI	FLAGS	imm8	%a	%b

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[u]	Defines if the operands are unsigned
11	[s]	Defines if the operation is SIMD (*)

3.2.3 divi

Division Immediate

divi %a, i8, %b

Computes $\%b = \%a/i8$.

0 7	8 13	14 19	20 25	26 31
OP_DIVI	FLAGS	imm8	%a	%b

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[u]	Defines if the operands are unsigned
11	[s]	Defines if the operation is SIMD (*)

3.2.4 mod

Modulo

mod %a, %b, %c

Computes $\%c = \%a mod \%b$.

0 7	8 13	14 19	20 25	26 31
OP_MOD	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[u]	Defines if the operands are unsigned
11	[s]	Defines if the operation is SIMD (*)

3.2.5 modi

Modulo Immediate

modi %a, i8, %b

Computes $\%b = \%a \; mod \; i8$.

0 7	8 13	14 19	20 25	26 31
OP_MODI	FLAGS	imm8	%a	%b

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[u]	Defines if the operands are unsigned
11	[s]	Defines if the operation is SIMD (*)

4 Logic

4.0.6 logic

Bitwise Logic

logic %a, %b, %c

Computes $\%c = f(\%a, \%b)$ where f is a logic function whose truth table is defined in the flags.

0 7	8 13	14 19	20 25	26 31
OP_LOGIC	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[01]	Defines the value of $f(0,0)$
11	[01]	Defines the value of $f(1,0)$
12	[01]	Defines the value of $f(0,1)$
13	[01]	Defines the value of $f(1,1)$

or is an alias for **logic.0111** .
and is an alias for **logic.0001** .
xor is an alias for **logic.0110** .
not is an alias for **logic.1010** .
nor is an alias for **logic.1000** .
nandis an alias for **logic.1110** .

5 Test

5.0.7 testxx

Test has been deprecated. Awaiting new definition.

Test on int values or float values.

testxx %a, %b, %c

Sets %c to 1 or 0 if the result of the test is true or false.

0 7	8 13	14 19	20 25	26 31
OP_TEST	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10-13	$[01]^4$	Defines the condiction to be tested (see table below)

testxx	xx	Value	Condition
testeq	eq	0000	%a is EQual to %b
testnq	nq	0001	%a is Not Equal to %b
testgt	gt	0010	%a is Greater Than %b
testge	ge	0011	%a is Greater than or Equal to %b
testlt	lt	0100	%a is Less Than %b
testle	le	0101	%a is Less than or Equal to %b
testcf	cf	0110	Carry flag is set
testzf	zf	0111	Zero flag is set

Table 1: Test Operations

6 Load/Store

7 Core Load/Store

7.0.8 load

Load

load [%a + %b * size], %c

$$\%c = [\%a + \%b * size]$$

0 7	8 13	14 19	20 25	26 31
OP_LOAD	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[e]	Defines the endianness little-endian if cleared (default) big-endian if set

7.0.9 store

Store

store %a, [%b + %c * size]

$[\%b + \%c * size] = \%a$

0 7	8 13	14 19	20 25	26 31
OP_STORE	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[e]	Defines the endianness little-endian if cleared (default) big-endian if set

7.0.10 mov

Move

mov [%a,] %b, %c

$if(\%a)\%c = \%b$

0 7	8 13	14 19	20 25	26 31
OP_MOV	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[n]	Defines if the data gets negated
11-12	[sz]	Defines how the high part of the destination register will be. (See table below)

Flag	Values	Function
(default)	00	High part remains unchanged
z	01	Zero extend
s	10	Sign extend
?	11	Reserved

7.0.11 loadcons

Load Constant

loadcons imm16, %a

$\%a = imm16$

Flags	Values	Function
8-9	[123]	Defines the shift parameter

0 7	8 9	10 25	26 31
OP_LOADCONS	EMPTY	imm16	%a

8 Optional Load/Store

8.0.12 loadi

Load Immediate

loadi [%a + imm10 * size], %b

$\%b = [\%a + i10 * size]$

0 7	8 9	10 19	20 25	26 31
OP_LOADI	FLAGS	imm10	%a	%b

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[e]	Defines the endianness little-endian if cleared (default) big-endian if set

8.0.13 storei

Store Immediate

storei %a, [%b + imm10 * size]

$[\%b + i10 * size] = \%a$

0 7	8 9	10 19	20 25	26 31
OP_STOREI	FLAGS	imm10	%a	%b

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[e]	Defines the endianness little-endian if cleared (default) big-endian if set

9 Internal registers

Get and Put internal register.

R/W	Description
R	Number of cycles
R	Number of cycles (countdown)
R	Number of instructions executed
R	Number of Pages Faults
R	Number of traps/interrupts
R	Number of FPU traps
R	Number of Cache hit/misses
R	Number of correct/incorrect branch predictions
R	Number of pipeline bubbles
R	Number of TLB hits/misses

Table 2: Performance Counters

R/W	Description
RW	Old Program Counter
RW	Old Machine Status Word
RW	Exception Vector
RW	Temporary
R	Exception Reason
R	Exception Number/Type

Table 3: Special Register for Exceptions

R/W	Description
R	Processor ID

Table 4: Diverse Special Registers

9.1 Core Internal registers

9.1.1 get

Get Internal Register

get IR[%a], %b

Get internal register at index %a and put its content in register %b.

0 7	8 19	20 25	26 31
OP_GET	EMPTY	%a	%b

9.1.2 put

PUT Internal Register

put %a, IR[%b]

Put contents of %a and put into internal register at index %b.

0 7	8 19	20 25	26 31
OP_PUT	EMPTY	%a	%b

9.2 Optional Internal Registers

9.2.1 geti

Get Internal Register Immediate

geti IR[imm16], %a

Get internal register at index imm16 and put its content in register %a.

0 7	8 9	10 25	26 31
OP_GETI	EMPTY	imm16	%a

9.2.2 puti

Put Internal Register Immediate

puti %a, IR[imm16]

Get internal register at index imm16 and put its content in register %a.

0 7	8 9	10 25	26 31
OP_PUTI	EMPTY	imm16	%a

10 Shift and Rotate

10.1 Core Shift and Rotate

10.1.1 shift

Shift

shift %a, %b, %c

Computes $\%c = \%a << \%b$ **or** $\%c = \%a >> \%b$

0 7	8 13	14 19	20 25	26 31
OP_SHIFT	**FLAGS**	**%a**	**%b**	**%c**

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[a]	Defines if the operand is signed
11	[lr]	Defines if the direction is left (cleared) or right (set).

10.1.2 rot

Rotation

rot %a, %b, %c

Computes $\%c = \%a < -\%b$ **or** $\%c = \%a- > \%b$

0 7	8 13	14 19	20 25	26 31
OP_ROT	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[a]	Defines if the operand is signed
11	[lr]	Defines if the direction is left (cleared) or right (set).

10.2 Optional Shift and Rotate

10.2.1 shifti

Shift Immediate

shifti %a, imm6, %b

Computes $\%b = \%a << imm6$ **or** $\%b = \%a << imm6$
imm6 is unsigned.

0 7	8 13	14 19	20 25	26 31
OP_SHIFTI	FLAGS	imm6	%a	%b

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[a]	Defines if the operand is signed
11	[lr]	Defines if the direction is left (cleared) or right (set).

10.2.2 roti

Rotate Immediate

roti %a, imm6, %b

Computes $\%b = \%a << imm6$ **or** $\%b = \%a >> imm6$
imm6 is unsigned.

0 7	8 13	14 19	20 25	26 31
OP_ROTI	FLAGS	imm6	%a	%b

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[a]	Defines if the operand is signed
11	[lr]	Defines if the direction is left (cleared) or right (set).

11 Floating Point Operations

11.1 Core Floating Point Operations

11.1.1 fadd

Floating Point Addition/Substraction

fadd %a, %b, %c

Computes $\%c = \%a + \%b$

0 7	8 13	14 19	20 25	26 31
OP_FADD	**FLAGS**	**%a**	**%b**	**%c**

Flags	Values	Function
8-9	**[f??]**	**Defines the size parameter**
10	**[n]**	**Defines if the second operand is negated**
11	**[s]**	**Defines if the operation is SIMD (*)**
12	**[x]**	**Defines if IEEE compliance isn't required (*)**

fsub %a, %b, %c

Computes $\%c = \%a - \%c$**.**
fsub is just an alias for **fadd.n** .

11.1.2 fmul

Floating Point Multiplication

fmul[f] %a, %b, C

Computes $\%c = \%a\%b$

0 7	8 13	14 19	20 25	26 31
OP_FMUL	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[f??]	Defines the size parameter
10	[n]	Defines if the second operand is negated
11	[s]	Defines if the operation is SIMD (*)
12	[x]	Defines if IEEE compliance isn't required (*)

11.1.3 fdiv

Floating Point Division

fdiv %a, %b, %c

Computes $\%c = \%a/\%b$

0 7	8 13	14 19	20 25	26 31
OP_FDIV	FLAGS	%a	%b	%c

Flags	Values	Function
8-9	[f??]	Defines the size parameter
10	[n]	Defines if the second operand is negated
11	[s]	Defines if the operation is SIMD (*)
12	[x]	Defines if IEEE compliance isn't required (*)

11.1.4 int2f and f2int

Integer to Floating Point and Floating Point to Integer

int2f %a, %b

f2int %a, %b

"int2f" converts integer number in register %a into a floating point number and put it in register %b.

"f2int" converts floating point number in register %a into an integer number and put it in register %b.

0 7	8 19	20 25	26 31
OP_FCONV	EMPTY	%a	%b

Flags	Values	Function
8-9	[f??]	Defines the size parameter
10	[n]	Defines if the second operand is negated
11	[s]	Defines if the operation is SIMD (*)
12	[x]	Defines if IEEE compliance isn't required (*)
13-15		Rounding modes see table below.

Rounding modes:

Value	Rounding mode
000	Nearest (default)
001	Towards 0
010	Away from 0
011	Towards $-\infty$
100	Towards $+\infty$

11.2 Optional Floating Point Operations

11.2.1 finv

Floating Point Inverse

finv %a, %b

Computes $\%b = \frac{1}{\%a}$

0 7	8 19	20 25	26 31
OP_FINV	EMPTY	%a	%b

Flags	Values	Function
8-9	[f??]	Defines the size parameter
10	[n]	Defines if the second operand is negated
11	[s]	Defines if the operation is SIMD (*)
12	[x]	Defines if IEEE compliance isn't required (*)

11.2.2 fsqrt

Floating Point Square Root

fsqrt %a, %b

Computes $\%b = \sqrt{\%a}$

0 7	8 19	20 25	26 31
OP_FSQRT	EMPTY	%a	%b

Flags	Values	Function
8-9	[f??]	Defines the size parameter
10	[n]	Defines if the second operand is negated
11	[s]	Defines if the operation is SIMD (*)
12	[x]	Defines if IEEE compliance isn't required (*)

11.2.3 finvsqrt

Floating Point Inverse Square Root

finvsqrt %a, %b

Computes $\%b = \frac{1}{\sqrt{\%a}}$

0 7	8 19	20 25	26 31
OP_FINVSQRT	EMPTY	%a	%b

Flags	Values	Function
8-9	[f??]	Defines the size parameter
10	[n]	Defines if the second operand is negated
11	[s]	Defines if the operation is SIMD (*)
12	[x]	Defines if IEEE compliance isn't required (*)

12 Branch

12.0.4 jmpa

Absolute Jump.

jmpa [%a,] %b

If %a contains a non-nil value jump to the address pointed by %b.

0 7	8 19	20 25	26 31
OP_JMPA	EMPTY	%a	%b

12.0.5 jmpi

Absolute Jump Immediate.

jmpi [%a,] %b, imm12

If %a contains a non-nil value jump to address pointed by %b+4*imm12.

0 7	8 19	20 25	26 31
OP_JMPI	imm12	%a	%b

12.0.6 jmpr

Relative Jump Immediate.

jmpr [%a,] imm18

The imm18 is a signed value. Warning! All code is aligned on a 32bit boundary so the imm18 value will be shifted to the left 2 times.

0 7	8 25	26 31
OP_JMPR	imm18	%a

13 CPU Control

13.0.7 syscall and trap

syscall [%a,] imm18

trap [%a,] imm18

Syscall are two names for the same instruction.

[FIX ME] But what does it do ?

The argument is ignored by the hardware and may be used to encode information for system software. To retrieve the argument system software must load the instruction word from memory.

0 7	8 25	26 31
OP_SYSTRAP	imm18	%a

13.0.8 halt

halt [%a,] [imm18]

Halts until an External Exception occurs

0 7	8 25	26 31
OP_HALT	imm18	%a

13.0.9 rfe

rfe [%a,] [imm18]

Return From Exception...
[FIX ME] Little short...

0 7	8 25	26 31
OP_RFE	imm18	%a

A Exception Handling (out of date)

Type 1	Software exception
Type 2	External exception (interrupt)
Type 3	Privilege Violation
Type 4	Memory Error
Type 5	Syscall

Table 5: Types of exceptions

[FIXME] How many differents types do we need to have, each one corresponding to one pointer in the exception vector (except for the hardware that takes all the rest).

Exception pointer vector has 64 entries. [FIXME] Confirm that.

SR_OPC	Old Program Counter
SR_OMSW	Old Machine Status Word
SR_EV	Exception Vector
SR_TMP	Temporary
SR_ER	Exception Reason
SR_ENT	Exception Number/Type

Table 6: Special Registers for Exception handling

1	External Interupt
2	Illegal Opcode
3	Malformed Instruction
4	Priviledge Violation
5	Integer divide by ZERO
6	FP divide by ZERO
7	FP INF-INF
8	FP INF/INF
9	FP ZERO/ZERO
10	FP ZERO*INF
11	FP SQRT(NEG)
12	Memory Exception

Table 7: Possible Exception Reasons Values

Upon the occurance of an exception, the proccessor performs the following..

- Invalidate or Flush Pipeline [FIXME]
- Store PC in SR_OPC
- Store MSW in SR_OMSW

- Switch(Exception Type)
  - Case Internal Exception: store Exception Type in SR_ENT
  - Case External Exception: store Exception Number in SR_ENT
  - Case Sycall: store the imm21 in SR_ENT
- Jump to the appropriate address using the Exception Vector(SR_EV).

Upon Call to the RFE instruction:

- Store SR_OPC in PC
- Store SR_OMSW in MSW

B Commments

New instruction formats (little-endian)

The distinction between Core and Optional functions