F-CPU Project **F-CPU**

Architecture Guide

by the F-CPU Design Team.

Draft 0.19991028

Contents

1 Foreword **4**

2 Instruction formats **4**
2.1 Format 0 . . . 4
2.2 Format 1 . . . 4
2.3 Format 2 . . . 4
2.4 Format 3 . . . 4
2.5 Size flags . . . 4

3 Data Manipulation **5**
3.1 Core Arithmetic . . . 5
3.1.1 add . . . 5
3.1.2 sub . . . 6
3.1.3 mul . . . 7
3.1.4 div . . . 8
3.2 Optional Arithmetic . . . 9
3.2.1 addi . . . 9
3.2.2 subi . . . 10
3.2.3 muli . . . 11
3.2.4 divi . . . 12
3.2.5 mod . . . 13
3.2.6 modi . . . 14
3.3 Core Shift and Rotate . . . 15
3.3.1 shift . . . 15
3.3.2 rot . . . 16
3.4 Optional Shift and Rotate . . . 17
3.4.1 shifti . . . 17
3.4.2 roti . . . 18
3.4.3 bitop . . . 19
3.5 Core Logic . . . 20
3.5.1 logic . . . 20
3.6 Optional SIMD Packing . . . 21
3.6.1 mix . . . 21
3.6.2 expand . . . 22
3.7 Floating Point Operations . . . 23
3.7.1 Introduction . . . 23
3.7.2 fadd . . . 24
3.7.3 fsub . . . 25
3.7.4 fmul . . . 26
3.7.5 fdiv . . . 27
3.7.6 int2f and f2int . . . 28
3.7.7 finv . . . 29
3.7.8 fsqrt . . . 30
3.7.9 finvsqrt . . . 31
3.8 Optional Misc. . . . 32
3.8.1 bitrev %r1, %r2, %r3 . . . 32
3.8.2 bitrevi %r1, imm6, %r3 . . . 33
3.8.3 byterev %r1, %r2 . . . 34

4 Load/Store **35**
4.1 Core Load/Store . . . 35
4.1.1 load . . . 35
4.1.2 store . . . 36
4.1.3 mov . . . 37
4.1.4 loadcons . . . 38
4.1.5 loadconsx . . . 39
4.1.6 cachemm . . . 41
4.2 Optional Load/Store . . . 42
4.2.1 loadi . . . 42
4.2.2 storei . . . 43
4.3 Internal registers info . . . 44
4.4 Core Internal registers . . . 45
4.4.1 get . . . 45
4.4.2 put . . . 46
4.5 Optional Internal Registers . . . 47
4.5.1 geti . . . 47
4.5.2 puti . . . 48

5 Flow Control **49**
5.1 Core Branch . . . 49
5.1.1 jmpa . . . 49
5.1.2 loadaddr . . . 50
5.1.3 loopentry . . . 51
5.1.4 loop . . . 52
5.2 Optional Branch . . . 53
5.2.1 jmpi . . . 53
5.2.2 jmpr . . . 54
5.3 Core CPU Control . . . 55
5.3.1 syscall and trap . . . 55
5.3.2 halt . . . 56
5.3.3 rfe . . . 57

A Exception Handling (out of date) **58**

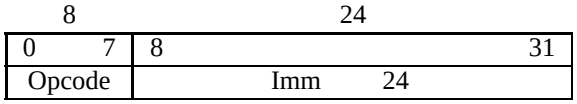
B Commments **60**

1 Foreword

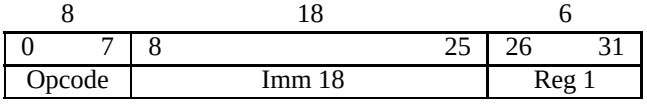
This is preliminary... Anything can be changed without notice.

2 Instruction formats

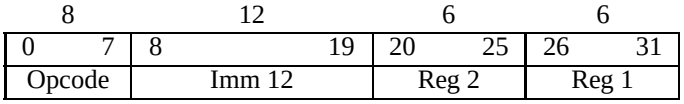
2.1 Format 0



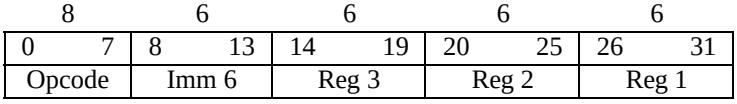
2.2 Format 1



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 Data Manipulation

3.1 Core Arithmetic

3.1.1 add

Addition

add %r1 , %r2, %r3

add performs an integer addition of the two source operands (%r1 and %r2) and puts the result in destination operand (%r3).

- The **size** flag indicates that **add** performs the addition on the whole operands or only on a part of the operands.
- The **SIMD** flag indicates that **add** performs multiple addition on parts of the operand (the size of these parts is defined by the **size** flags).
- The **saturate** flag indicates that **add** does not "wrap" if the result is bigger than the size of the operands.

0 7	8 13	14 19	20 25	26 31
OP_ADD	FLAGS	%r1	%r2	%r3

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[s]	Defines if the operation is SIMD (*)
12	[m]	Defines if the operation should saturate
13		Unused.

3.1.2 sub

Substraction

sub %r1, %r2, %r3

sub performs an integer substraction of the two source operands (%r1 and %r2) and puts the result in destination operand (%r3).

- The **size** flag indicates that **sub** performs the substraction on the whole operands or only on a part of the operands.
- The **SIMD** flag indicates that **sub** performs multiple substraction on parts of the operand (the size of these parts is defined by the **size** flags).
- The **floor** flag indicates that **sub** does not "wrap" if the second operand is bigger than the first one.

0 7	8 13	14 19	20 25	26 31
OP_SUB	FLAGS	%r1	%r2	%r3

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[s]	Defines if the operation is SIMD (*)
12	[f]	Defines if the operation should floor
13		Unused.

3.1.3 mul

Multiplication

mul %r1, %r2, %r3

mul performs an integer multiplication of the two source operands (%r1 and %r2) and puts the result in destination operand (%r3).

- The **size** flag indicates that **mul** performs the multiplication on the whole operands or only on a part of the operands. It only puts the lower part of the result.
- The **SIMD** flag indicates that **mul** performs multiple multiplication on parts of the operand (the size of these parts is defined by the **size** flags).

0 7	8 13	14 19	20 25	26 31
OP_MUL	FLAGS	%r1	%r2	%r3

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

3.1.4 div

Division

div %r1, %r2, %r3

div performs an integer division of the two source operands (%r1 and %r2) and puts the result in destination operand (%r3).

- The **size** flag indicates that **div** performs the division on the whole operands or only on a part of the operands. It only puts the lower part of the result.
- The **SIMD** flag indicates that **div** performs multiple division on parts of the operand (the size of these parts is defined by the **size** flags).

0 7	8 13	14 19	20 25	26 31
OP_DIV	FLAGS	%r1	%r2	%r3

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

3.2 Optional Arithmetic

3.2.1 addi

Addition Immediate

addi %r1, i8, %r2

Compute $\%r2 = \%r1 + i8$.

0 7	8 11	12 19	20 25	26 31
OP_ADDI	FLAGS	imm8	%r1	%r2

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

3.2.2 subi

Substraction Immediate

subi %r1, i8, %r2

Computes $\%r2 = \%r1 - i8$

0 7	8 11	12 19	20 25	26 31
OP_SUBI	FLAGS	imm8	%r1	%r2

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

3.2.3 muli

Multiplication Immediate

muli %r1, i8, %r2

Computes $\%r2 = \%r1 \times i8$.

0 7	8 11	12 19	20 25	26 31
OP_MULI	FLAGS	imm8	%r1	%r2

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

3.2.4 divi

Division Immediate

divi %r1, i8, %r2

Computes $\%r2 = \%r1/i8$.

0 7	8 11	12 19	20 25	26 31
OP_DIVI	FLAGS	imm8	%r1	%r2

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

3.2.5 mod

Modulo

mod %r1, %r2, %r3

Computes $\%r3 = \%r1 mod \%r2$.

0 7	8 13	14 19	20 25	26 31
OP_MOD	FLAGS	%r1	%r2	%r3

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

3.2.6 modi

Modulo Immediate

modi %r1, i8, %r2

Computes $\%r2 = \%r1 \; mod \; i8$.

0 7	8 11	12 19	20 25	26 31
OP_MODI	FLAGS	imm8	%r1	%r2

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

3.3 Core Shift and Rotate

3.3.1 shift

Shift

shift %r1, %r2, %r3

Computes $\%r3 = \%r1 << \%r2$ or $\%r3 = \%r1 >> \%r2$

0 7	8 13	14 19	20 25	26 31
OP_SHIFT	FLAGS	%r1	%r2	%r3

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

3.3.2 rot

Rotation

rot %r1, %r2, %r3

Computes $\%r3 = \%r1 < -\%r2$ or $\%r3 = \%r1 -> \%r2$

0 7	8 13	14 19	20 25	26 31
OP_ROT	FLAGS	%r1	%r2	%r3

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

3.4 Optional Shift and Rotate

3.4.1 shifti

Shift Immediate

shifti %r1, imm6, %r2

Computes $\%r2 = \%r1 << imm6$ or $\%r2 = \%r1 << imm6$
imm6 is unsigned.

0 7	8 13	14 19	20 25	26 31
OP_SHIFTI	FLAGS	imm6	%r1	%r2

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

3.4.2 roti

Rotate Immediate

roti %r1, imm6, %r2

Computes $\%r2 = \%r1 < -imm6$ or $\%r2 = \%r1 -> imm6$
imm6 is unsigned.

0 7	8 13	14 19	20 25	26 31
OP_ROTI	FLAGS	imm6	%r1	%r2

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

3.4.3 bitop

Single Bit Operation

bitop %r1, imm6, %r2

Apply a logical operation to %r1 and bit imm6 set.

0 7	8 13	14 19	20 25	26 31
OP_BITOP	FLAGS	imm6	%r1	%r2

Flags	Values	Function
8-9	[scxt]	Defines the operation

bset is an alias for **bitop.s** (bit set, or).
bclear is an alias for **bitop.c** (bit clear, andn).
bchange is an alias for **bitop.x** (bit change, xor).
btest is an alias for **bitop.t** (bit test, and).

3.5 Core Logic

3.5.1 logic

Bitwise Logic

logic %r1, %r2, %r3

Computes $\%r3 = f(\%r1, \%r2)$ 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	%r1	%r2	%r3

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** .
Remark: XOR should be used to compare two numbers for equality

3.6 Optional SIMD Packing

3.6.1 mix

mix %r1, %r2, %r3

Mix %r1 and %r2 into %r3.

0 7	8 13	14 19	20 25	26 31
OP_MIX	FLAGS	%r1	%r2	%r3

Flags	Values	Function
8	[hl]	Defines which part of the words should be mixed (high, low).

3.6.2 expand

expand %r1, %r2

Expand %r1 into %r2.

0 7	8 19	20 25	26 31
OP_EXPAND	EMPTY	%r1	%r2

Flags	Values	Function
8	[hl]	Defines which in part of the word the result should be put (high, low).

3.7 Floating Point Operations

3.7.1 Introduction

There are different levels of implementation of floating point operations.

Level	Instructions implemented
Level 0	No FP
Level 1	fadd, fsub, fmul, finv_app, sqrt_inv_app
Level 2	fadd, fsub, fmul, finv, sqrt
Level 2	fadd, fsub, fmul, div, finv, sqrt, sqrt_inv

3.7.2 fadd

Floating Point Addition

fadd %r1, %r2, %r3

fadd performs a floating addition of the two source operands (%r1 and %r2) and puts the result in destination operand (%r3). The operation should be IEEE-754 compliant.

- The **size** flag indicates that **fadd** performs the addition on the whole operands or only on a part of the operands. This size flags is different from the integer size flag: only two values are currently assigned (01) for 64 bits and (00) for 32 bits.
- The **SIMD** flag indicates that **fadd** performs multiple addition on parts of the operand (the size of these parts is defined by the **size** flags).
- The **Exception** flag indicates that **fadd** should generate exceptions (when needed) in accordance to the IEEE-754 standard.

0 7	8 13	14 19	20 25	26 31
OP_FADD	FLAGS	%r1	%r2	%r3

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

3.7.3 fsub

Floating Point Substraction

fsub %r1, %r2, %r3

fsub performs a floating substraction of the two source operands (%r1 and %r2) and puts the result in destination operand (%r3). The operation should be IEEE-754 compliant.

- The **size** flag indicates that **fsub** performs the substraction on the whole operands or only on a part of the operands. This size flags is different from the integer size flag: only two values are currently assigned (01) for 64 bits and (00) for 32 bits.
- The **SIMD** flag indicates that **fsub** performs multiple substraction on parts of the operand (the size of these parts is defined by the **size** flags).
- The **exception** flag indicates that **fsub** should generate exceptions (when needed) in accordance to the IEEE-754 standard.

0 7	8 13	14 19	20 25	26 31
OP_FSUB	FLAGS	%r1	%r2	%r3

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

3.7.4 fmul

Floating Point Multiplication

fmul[f] %r1, %r2, C

fmul performs a floating multiplication of the two source operands (%r1 and %r2) and puts the result in destination operand (%r3). The operation should be IEEE-754 compliant.

- The **size** flag indicates that **fmul** performs the multiplication on the whole operands or only on a part of the operands. This size flags is different from the integer size flag: only two values are currently assigned (01) for 64 bits and (00) for 32 bits.
- The **SIMD** flag indicates that **fmul** performs multiple multiplication on parts of the operand (the size of these parts is defined by the **size** flags).
- The **exception** flag indicates that **fmul** should generate exceptions (when needed) in accordance to the IEEE-754 standard.

0 7	8 13	14 19	20 25	26 31
OP_FMUL	FLAGS	%r1	%r2	%r3

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

3.7.5 fdiv

Floating Point Division

fdiv %r1, %r2, %r3

fdiv performs a floating division of the two source operands (%r1 and %r2) and puts the result in destination operand (%r3). The operation should be IEEE-754 compliant.

- The **size** flag indicates that **fdiv** performs the division on the whole operands or only on a part of the operands. This size flags is different from the integer size flag: only two values are currently assigned (01) for 64 bits and (00) for 32 bits.
- The **SIMD** flag indicates that **fdiv** performs multiple division on parts of the operand (the size of these parts is defined by the **size** flags).
- The **exception** flag indicates that **fdiv** should generate exceptions (when needed) in accordance to the IEEE-754 standard.

0 7	8 13	14 19	20 25	26 31
OP_FDIV	FLAGS	%r1	%r2	%r3

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

3.7.6 int2f and f2int

Integer to Floating Point and Floating Point to Integer

int2f %r1, %r2

f2int %r1, %r2

"int2f" converts integer number in register %r1 into a floating point number and put it in register %r2.

"f2int" converts floating point number in register %r1 into an integer number and put it in register %r2.

0 7	8 19	20 25	26 31
OP_FCONV	EMPTY	%r1	%r2

Flags	Values	Function
8-9	[f??]	Defines the size parameter
10		Direction flag.
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 $+\infty$

3.7.7 finv

Floating Point Inverse

finv %r1, %r2

Computes $\%r2 = \frac{1}{\%r1}$

0 7	8 19	20 25	26 31
OP_FINV	EMPTY	%r1	%r2

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

3.7.8 fsqrt

Floating Point Square Root

fsqrt %r1, %r2

Computes $\%r2 = \sqrt{\%r1}$

0 7	8 19	20 25	26 31
OP_FSQRT	EMPTY	%r1	%r2

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

3.7.9 finvsqrt

Floating Point Inverse Square Root

finvsqrt %r1, %r2

Computes $\%r2 = \frac{1}{\sqrt{\%r1}}$

0 7	8 19	20 25	26 31
OP_FINVSQRT	EMPTY	%r1	%r2

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

3.8 Optional Misc.

3.8.1 bitrev %r1, %r2, %r3

Reverses the bits from %r1 and shifts the result to the right %r2 bits and put the result in %r3.

0 7	8 13	14 19	20 25	26 31
OP_BITREV	FLAGS	%r1	%r2	%r3

3.8.2 bitrevi %r1, imm6, %r3

Reverses the bits from %r1 and shifts the result to the right imm6 bits and put the result in %r3.

0 7	8 13	14 19	20 25	26 31
OP_BITREVI	FLAGS	imm6	%r1	%r2

3.8.3 byterev %r1, %r2

Reverse bytes from the source operand %r1 (change the endianism) and put the result into %r2.

0 7	8 19	20 25	26 31
OP_BYTEREV	EMPTY	%r1	%r2

Flags	Values	Function
8-9	[qdb]	Defines the size parameter

4 Load/Store

4.1 Core Load/Store

4.1.1 load

Load

load [%r1 + %r2 * size], %r3

$\%r3 = [\%r1 + \%r2 * size]$

0 7	8 13	14 19	20 25	26 31
OP_LOAD	FLAGS	%r1	%r2	%r3

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[e]	Defines the endianness little-endian if cleared (default) big-endian if set
11-13		RESERVED

4.1.2 store

Store

store %r1, [%r2 + %r3 * size]

$[\%r2 + \%r3 * size] = \%r1$

0 7	8 13	14 19	20 25	26 31
OP_STORE	FLAGS	%r1	%r2	%r3

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[e]	Defines the endianness little-endian if cleared (default) big-endian if set
11-13		RESERVED

4.1.3 mov

Move

mov [%r1,] %r2, %r3

$if(\%r1)\%r3 = \%r2$

0 7	8 13	14 19	20 25	26 31
OP_MOV	FLAGS	%r1	%r2	%r3

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10-11	[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

Remark: move %r0, %r0, %r0 is an alias for NOP.

4.1.4 loadcons

Load Constant

loadcons imm16, %r1

Loads the imm16 constant into the register %r1 at the specified location (shifts of 16 bits). The rest of the register remains unmodified.

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

0 7	8 9	10 25	26 31
OP_LOADCONS	EMPTY	imm16	%r1

4.1.5 loadconsx

Load Constant with Sign Extension

loadconsx imm16, %r1

Loads the imm16 constant into the register %r1 at the specified location (shifts of 16 bits). The higher part of the register is assigned the value of the most significant bit of the constant. The lower part of the register remains unmodified.

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

0 7	8 9	10 25	26 31
OP_LOADCONSX	EMPTY	imm16	%r1

```
/*
  LOADCONST.C by WHYGEE  14 septembre 1999

  to be included in a compiler or an assembler, after some
  interface fixing : it currently outputs to stderr, it will
  output to a file the same way.
*/

#include "stdlib.h"
#include "stdio.h"

#define MAXSIZE (sizeof(long long int))
/* should be ideally 8 */

void emit_constant(unsigned long long int c, unsigned char reg)
{
  unsigned short int data[MAXSIZE>>1];
  signed long long int t,u;
  signed int s=0;

  if (reg==0)
    {
      fprintf(stderr,"\\n Error : can't write to register 0 \\n");
      exit(-1);  /* should be performed by an error routine that does this cleanly */
    }
  if (c==0)
    {
      fprintf(stderr,"mov r\%d,r0\\n",reg);
    }
  else if (c==-1)
    {
      fprintf(stderr,"logic.1111 r\%d,r0,r0\\n",reg);
    }
  else if ((c>65535)&((c & -c)==c))
    /* a power of two, but the latency of bitset is higher */
    {
      do { s++; c>>=1; } while (c!=0); /* find the LSB */
      if (s>63)
        {
          fprintf(stderr,"loadconsts r\%d,0x\%04X\\n",reg,s);
          fprintf(stderr,"bitset r\%d,r0,r\%d\\n",reg,reg);
        }
      else
        {
          fprintf(stderr,"bitset r\%d,r0,\%d\\n",reg,s);
        }
    }
  else /* any kind of number */
    {
      u=c;
      do {
        t=u;
        data[s]=t & 0xFFFF;
        u=t>>16;
        s++;
      } while ((t!=u) & (s<MAXSIZE>>1));

      s--;

      /* handle the case where the MSB of the highest data is not the sign */
      if ((data[s]^data[s-1])& 0x8000)
        {
          fprintf(stderr,"loadconsts.\%d r\%d,0x\%04X\\n", s,reg,data[s]);
          s--;
          fprintf(stderr,"loadconst.\%d r\%d,0x\%04X\\n", s,reg,data[s]);
          s--;
        }
      else
        {
          s--;
          fprintf(stderr,"loadconsts.\%d r\%d,0x\%04X\n", s,reg,data[s]);
          s--;
        }

      while (s>=0)
        {
          fprintf(stderr,"loadconst.\%d r\%d,0x\%04X\\n", s,reg,data[s]);
          s--;
        }
    }
}
```

4.1.6 cachemm

Cache Memory Management

prefetch, flush a data block to/from a memory level.

cachemm %r1, %r2

Flags	Values	Function
8-9	[qdb]	Defines the size parameter
10	[fp]	Prefetch/Fetch
11	[l]	Lock. This flag means that the data are static and will be used a lot
12-14	$[01]^3$	Memory level (see table below)

D	000	data L1 cache
I	001	instructions L1 cache
C	010	onchip unified cache
	011	[unused]
U	100	offchip unified cache
L	101	local memory
G	110	global memory
V	111	virtual memory (hard disk)

0 7	8 19	20 25	26 31
OP_CACHEMM	EMPTY	%r1	%r2

[subject to changes as discussions go] also possible: ask for compression on the fly.

example : ”flushg ra,rb” flushes rb bytes starting at address ra from every memory level until global memory. Any cache (L1, L2, local...) containing data that belong to the block is updated in main memory and the corresponding cache spaces are freed (available for future use). this should be executed everytime the programer knows that he won’t use a block of data until a certain moment, and the cache level is a hint for performance.

”preftchu ra,rb” copies the data block at address ra and size rb that is present in lower memory levels (virtual, global, local) to the unified offchip memory (at least).

forms : rr or ri (size could be immediate)

These instructions are very important for memory management, and should be used when performing SMC (for memory coherency).

4.2 Optional Load/Store

4.2.1 loadi

Load Immediate

loadi [%r1 + imm9 * size], %r2

$\%r2 = [\%r1 + imm9 * size]$

0 7	8 10	11 19	20 25	26 31
OP_LOADI	FLAGS	imm9	%r1	%r2

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

4.2.2 storei

Store Immediate

storei %r1, [%r2 + imm9 * size]

$[\%r2 + imm9 * size] = \%r1$

0 7	8 10	11 19	20 25	26 31
OP_STOREI	FLAGS	imm9	%r1	%r2

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

4.3 Internal registers info

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 1: 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 2: Special Register for Exceptions

R/W	Description
R	Processor ID

Table 3: Diverse Special Registers

4.4 Core Internal registers

4.4.1 get

Get Internal Register

get IR[%r1], %r2

Get internal register at index %r1 and put its content in register %r2. The whole register gets dumped. There is no size flag.

0 7	8 19	20 25	26 31
OP_GET	EMPTY	%r1	%r2

4.4.2 put

PUT Internal Register

put %r1, IR[%r2]

Put contents of %r1 and put into internal register at index %r2. The whole register gets dumped. There is no size flag.

0 7	8 19	20 25	26 31
OP_PUT	EMPTY	%r1	%r2

4.5 Optional Internal Registers

4.5.1 geti

Get Internal Register Immediate

geti IR[imm16], %r1

Get internal register at index imm16 and put its content in register %r1. The whole register gets dumped. There is no size flag.

0 7	8 9	10 25	26 31
OP_GETI	EMPTY	imm16	%r1

4.5.2 puti

Put Internal Register Immediate

puti %r1, IR[imm16]

Get internal register at index imm16 and put its content in register %r1. The whole register gets dumped. There is no size flag.

0 7	8 9	10 25	26 31
OP_PUTI	EMPTY	imm16	%r1

5 Flow Control

5.1 Core Branch

5.1.1 jmpa

Absolute Jump.

jmpa [%r1,] %r2

If %r1 contains a non-nil value jump to the address pointed by %r2.

Flags	Values	Function
8	[n]	Negates the condition
9-10	[lm]	Test the MSB or the LSB

0 7	8 19	20 25	26 31
OP_JMPA	EMPTY	%r1	%r2

5.1.2 loadaddr

Load Address

loadaddr imm18 %r1

Stores $PC + imm18$ into %r1.
The result is a 64 bit address. imm18 is a signed value.

0 7	8 25	26 31
OP_LOADADDR	imm18	%r1

5.1.3 loopentry

Loop Entry

loopentry %r1

Stores $PC + 4$ into %r1.

0 7	8 25	26 31
OP_LOOPENTRY	EMPTY	%r1

This instruction is a special form of loadaddr.

5.1.4 loop

Loop

loop %r1, %r2

Performs two parallel things :

- decrements %r2
- checks if the old value of %r2 was zero. In the case of non nullity, it branches to the address contained in %r1.

This overlapping of the operations allows greater parallelism and lower latency : we can loop fast without compromising security.

0 7	8 19	20 25	26 31
OP_LOOP	EMPTY	%r1	%r2

5.2 Optional Branch

5.2.1 jmpi

Absolute Jump Immediate.

jmpi [%r1,] %r2, imm12

If %r1 contains a non-nil value jump to address pointed by %r2+4*imm12.

0 7	8 19	20 25	26 31
OP_JMPI	imm12	%r1	%r2

5.2.2 jmpr

Relative Jump Immediate.

jmpr [%r1,] 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	%r1

5.3 Core CPU Control

5.3.1 syscall and trap

syscall [%r1,] imm18

trap [%r1,] 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	%r1

5.3.2 halt

halt [%r1,] [imm18]

Halts until an External Exception occurs

0 7	8 25	26 31
OP_HALT	imm18	%r1

5.3.3 rfe

rfe [%r1,] [imm18]

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

0 7	8 25	26 31
OP_RFE	imm18	%r1

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 4: 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 5: 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 6: 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