

Lemberg

Wolfgang Puffitsch

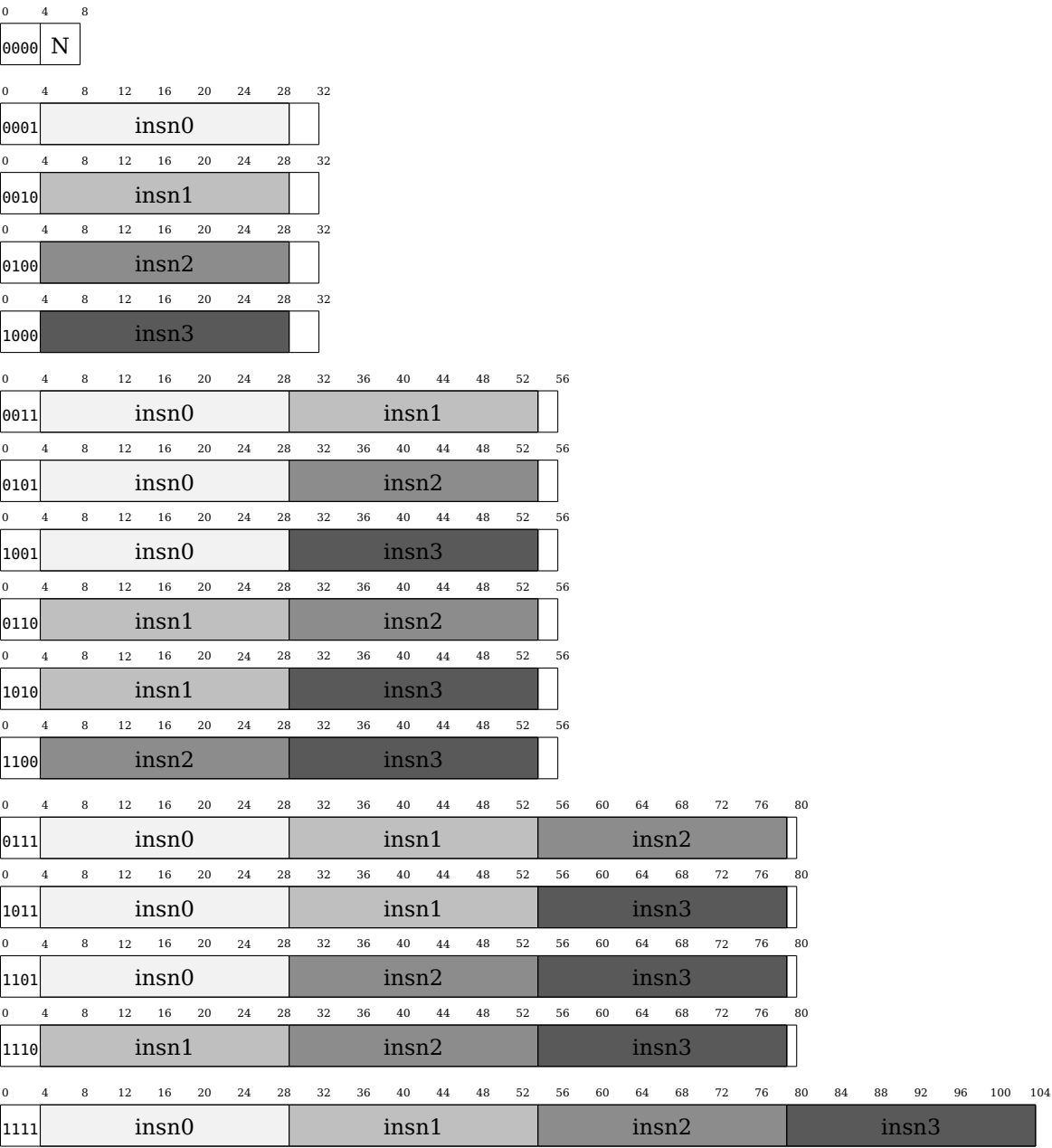
- Why do you pronounce VLIW with an “F” at the end?
 - Because I also pronounce Lviv that way.

Contents

1 Opcode Formats	2
1.1 Bundle Formats	2
1.2 Instruction Formats	3
2 Register File	5
2.1 General-Purpose Registers	5
2.2 Special Registers	6
3 Operations	7
3.1 Bit/Byte/Half-word Operations	9
3.2 Compare Operations	9
3.3 Bit Test	9
3.4 Flag Combination Operations	9
3.5 Branch Zero Operations	10
3.6 Jump Operations	10
3.7 Floating-Point Operations	11
3.7.1 Floating-Point Comparison	12
4 Default Memory Map	13
5 Notes	13

1 Opcode Formats

1.1 Bundle Formats



1.2 Instruction Formats

- Base format **B**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
opCode						srcReg1						srcReg2						destReg			0	c	F		
opCode						srcReg						imm						destReg			1	c	F		

- Comparison format **C**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25		
opCode						srcReg1						srcReg2						op		dest		0	c		F		
opCode						srcReg						imm						op		dest		1	c		F		

- Flag combination format **X**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
opCode						–	d	–	s1	–	s2	i1	i2	op	c	F									

- Floating-point format **F**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
opCode						dest						src1						src2			op			c	F

- Global address format **G**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
opCode						address																			

- Global address load format **H**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
opC.			dest			address																			

dest values 000-011 address r0-r3, values 100-111 address r16-r18.

- Immediate format **I**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
opCode						destReg						imm										c	F		

- Branch format **J**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
opCode						offset															d	c	F		

- Branch compare zero format **Z**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
opCode						src						offset									d	op			

- Load format **L**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
opCode						addrReg						offset										c	F		

- Store format **S**:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
opCode						addrReg						valReg				offset					0	c	F		
opCode						addrReg						imm				offset					1	c	F		

c ... condition: 1 ... if true, 0 ... if false

F ... condition flag to use

d ... delayed branch

2 Register File

2.1 General-Purpose Registers

Index	Name	Description	Use
0	r0	global reg 0	integer result 0, argument 0
1	r1	global reg 1	integer result 1, argument 1
2	r2	global reg 2	argument 2
3	r3	global reg 3	argument 3
4	r4	global reg 4	temporary
5	r5	global reg 5	temporary
6	r6	global reg 6	temporary
7	r7	global reg 7	temporary
8	r8	global reg 8	temporary
9	r9	global reg 9	temporary
10	r10	global reg 10	callee saved
11	r11	global reg 11	callee saved
12	r12	global reg 12	callee saved
13	r13	global reg 13	callee saved
14	r14	global reg 14	frame pointer, callee saved
15	r15	global reg 15	stack pointer
16	rn.16	local reg 0	temporary
17	rn.17	local reg 1	temporary
18	rn.18	local reg 2	temporary
19	rn.19	local reg 3	temporary
20	rn.20	local reg 4	temporary
21	rn.21	local reg 5	temporary
22	rn.22	local reg 6	temporary
23	rn.23	local reg 7	callee saved
24	rn.24	local reg 8	callee saved
25	rn.25	local reg 9	callee saved
26	rn.26	local reg 10	callee saved
27	rn.27	local reg 11	callee saved
28	rn.28	local reg 12	callee saved
29	rn.29	local reg 13	callee saved
30	rn.30	local reg 14	reserved
31	r31	memory load result	

2.2 Special Registers

Index	Name	Description	Use
0	\$c0	Condition flag 0	global, always true
1	\$c1	Condition flag 1	global
2	\$c2	Condition flag 2	global
3	\$c3	Condition flag 3	global
4	\$muln.0	Multiplication result 0	per-cluster
5	\$muln.1	Multiplication result 1	per-cluster
6	\$rb	Return base	global
7	\$ro	Return offset	global
8	\$ba	Base address	read only, global
9	\$irb	Interrupt return base	global
10	\$iro	Interrupt return offset	global
11	\$itmp	Interrupt temporary	old value r31 at start of interrupt, global
12	?		
13	?		
14	?		
15	?		
16	\$f0, \$d0	FPU register 0	floating-point result 0
17	\$f1	FPU register 1	floating-point result 1
18	\$f2, \$d1	FPU register 2	temporary
19	\$f3	FPU register 3	temporary
20	\$f4, \$d2	FPU register 4	temporary
21	\$f5	FPU register 5	temporary
22	\$f6, \$d3	FPU register 6	temporary
23	\$f7	FPU register 7	temporary
24	\$f8, \$d4	FPU register 8	temporary
25	\$f9	FPU register 9	temporary
26	\$f10, \$d5	FPU register 10	callee saved
27	\$f11	FPU register 11	callee saved
28	\$f12, \$d6	FPU register 12	callee saved
29	\$f13	FPU register 13	callee saved
30	\$f14, \$d7	FPU register 14	callee saved
31	\$f15	FPU register 15	callee saved

3 Operations

Opcode	Name	Fmt	Unit	Semantics
Arithmetic				
00 0000	add	B	A	dest = src1 + src2
00 0001	sub	B	A	dest = src1 - src2
00 0010	addi	I	A	dest += src1
00 0011	s1add	B	A	dest = src1 + src2*2
00 0100	s2add	B	A	dest = src1 + src2*4
00 0101	and	B	A	dest = src1 & src2
00 0110	or	B	A	dest = src1 src2
00 0111	xor	B	A	dest = src1 ^ src2
00 1000	sl	B	A	dest = src1 << src2
00 1001	sr	B	A	dest = src1 >>> src2
00 1010	sra	B	A	dest = src1 >> src2
00 1011	rl	B	A	dest = (src1 << src2) (src1 >>> (32-src2))
00 1100	mul	B	A	\$mul = src1 * src2
00 1101	carr	B	A	dest = ((uint64_t)src1+(uint64_t)src2)>>>32
00 1110	borr	B	A	dest = ((uint64_t)src1-(uint64_t)src2)>>>32
00 1111	bbh	B	A	bit/byte/half-word operation (see Section 3.1)
—				
010 000	?			
010 001	?			
010 010	?			
010 011	?			
Conditions				
010 100	cmp	C	A	dest = src1 op src2, signed, (see Section 3.2)
010 101	cmpl	C	A	dest = src1 op src2, unsigned, (see Section 3.2)
010 110	btest	C	A	dest = (src1 & (1 << src)) op 0 (see Section 3.3)
010 111	comb	X	A	flag combination operation (see Section 3.4)
Constants				
0110 00	ldi	I	A	dest = imm, signed
0110 01	ldiu	I	A	dest = imm, unsigned
0110 10	ldim	I	A	dest = imm << 11, signed
0110 11	ldih	I	A	dest = imm << 21
Flow Control				
0111 00	br	J	J	pc = pc+offset
0111 01	brz	Z	J	if (src op 0) pc = pc+offset (see Section 3.5)
0111 10	jop	B	J,M	jump operation (see Section 3.6)
0111 11	callg	G	J,M	\$rb = \$ba, \$ro = pc, \$ba = addr*4, pc = 0

Memory Accesses				
10 0000	stm.a	S	M	[addr+offset*4] = val, all caches, int32_t
10 0001	stmh.a	S	M	[addr+offset*2] = val, all caches, int16_t
10 0010	stmb.a	S	M	[addr+offset] = val, all caches, int8_t
10 0011	stm.s	S	M	[addr+offset*4] = val, stack cache, int32_t
10 0100	stmh.s	S	M	[addr+offset*2] = val, stack cache, int16_t
10 0101	stmb.s	S	M	[addr+offset] = val, stack cache, int8_t
10 0110	ldm.b	L	M	issue r31 = [addr+offset], bypass caches
10 0111	ldm.d	L	M	issue r31 = [addr+offset], direct mapped cache
10 1000	ldm.f	L	M	issue r31 = [addr+offset], fully assoc. cache
10 1001	ldm.s	L	M	issue r31 = [addr+offset], stack cache
10 1010	ldmg.d	G	M	issue r31 = [addr*4], direct mapped cache
10 1011	ldmr.f	S	M	issue r31 = [addr+(val<<offset)], fully assoc. cache
Specials				
1011 00	ldx	B	A	dest = src1, src1 refers to special register
1011 01	stx	B	A	dest = src1, dest refers to special register
1011 10	fop	F	F	floating-point operation (see Section 3.7)
1011 11	wb.s	L	M	write back data from stack cache
Global Address Constants				
110	ldga	H	A	dest = address*4, unsigned
—				
111 000	?			
111 001	?			
111 010	?			
111 011	?			
111 100	?			
111 101	?			
111 110	?			
111 111	?			

3.1 Bit/Byte/Half-word Operations

Src2	Name	Semantics
Sub-Word Extraction		
000 00	sext8	dest = (int8_t)src1
000 01	sext16	dest = (int16_t)src1
000 10	zext8	dest = (uint8_t)src1
000 11	zext16	dest = (uint16_t)src1
Bit Counting		
001 00	clz	count leading zeros
001 01	ctz	count trailing zeros
001 10	pop	count ones
001 11	par	compute parity
Sub-Word Memory Loads		
010 00	msext8	dest = (int8_t)(r31 >> 8*(src1 & 3))
010 01	msext16	dest = (int16_t)(r31 >> 8*(src1 & 2))
010 10	mzext8	dest = (uint8_t)(r31 >> 8*(src1 & 3))
010 11	mzext16	dest = (uint16_t)(r31 >> 8*(src1 & 2))

3.2 Compare Operations

Op	Name	Semantics
000	eq	dest = (src1 == src2)
001	ne	dest = (src1 != src2)
010	lt	dest = (src1 < src2)
011	ge	dest = (src1 >= src2)
100	le	dest = (src1 <= src2)
101	gt	dest = (src1 > src2)

3.3 Bit Test

Op	Semantics
--0	dest = (src1 & (1 << src)) != 0)
--1	dest = (src1 & (1 << src)) == 0)

3.4 Flag Combination Operations

Op	Name	Semantics
00	and	d = (i1 ^ s1) & (i2 ^ s2)
01	or	d = (i1 ^ s1) (i2 ^ s2)
10	xor	d = (i1 ^ s1) ^ (i2 ^ s2)

3.5 Branch Zero Operations

Op	Name	Semantics
000	eq	if (src == 0) pc = pc+offset
001	ne	if (src != 0) pc = pc+offset
010	lt	if (src < 0) pc = pc+offset
011	ge	if (src >= 0) pc = pc+offset
100	le	if (src <= 0) pc = pc+offset
101	gt	if (src > 0) pc = pc+offset

3.6 Jump Operations

Src2	Name	Semantics
000 00	brind	pc = src1
000 01	call	\$rb = \$ba, \$ro = pc, \$ba = src1, pc = 0
000 10	ret	\$ba = \$rb, pc = \$ro
000 11	iret	\$ba = \$irb, pc = \$iro

3.7 Floating-Point Operations

Op	Src2	Src1	Name	Semantics
0000	-	-	fadd	dest = src1 + src2, single
0001	-	-	fsub	dest = src1 - src2, single
0010	-	-	fmul	dest = src1 * src2, single
0011	-	-	fmac	dest += src1 * src2, single
0100	-	-	dadd	dest = src1 + src2, double
0101	-	-	dsub	dest = src1 - src2, double
0110	-	-	dmul	dest = src1 * src2, double
0111	-	-	dmac	dest += src1 * src2, double
1000	-	-	fcmp	comparison, single \rightarrow int32_t (see Section 3.7.1)
1001	-	-	dcmp	comparison, double \rightarrow int32_t (see Section 3.7.1)
1010	-	-	?	
1011	-	-	?	
1100	-	-	?	
1101	-	-	?	
1110	-	-	?	
1111	0000	-	fmov	dest = src1, single
1111	0001	-	fneg	dest = -src1, single
1111	0010	-	fabs	dest = abs(src1), single
1111	0011	0000	fzero	dest = 0.0, single
1111	0011	0001	fhalf	dest = 0.5, single
1111	0011	0010	fone	dest = 1.0, single
1111	0011	0011	ftwo	dest = 2.0, single
1111	0011	1111	fnan	dest = NaN, single
1111	0100	-	dmov	dest = src1, double
1111	0101	-	dneg	dest = -src1, double
1111	0110	-	dabs	dest = abs(src1), double
1111	0111	0000	dzero	dest = 0.0, double
1111	0111	0001	dhalf	dest = 0.5, double
1111	0111	0010	done	dest = 1.0, double
1111	0111	0011	dtwo	dest = 2.0, double
1111	0111	1111	dnan	dest = NaN, double
1111	1000	-	rnd	dest = (float)src1, double \rightarrow single
1111	1001	-	ext	dest = (double)src1, single \rightarrow double
1111	1010	-	si2sf	dest = (float)src1, int32_t \rightarrow single
1111	1011	-	si2df	dest = (double)src1, int32_t \rightarrow double
1111	1100	-	sf2si	dest = (int)src1, single \rightarrow int32_t
1111	1101	-	df2si	dest = (int)src1, double \rightarrow int32_t

3.7.1 Floating-Point Comparison

Result Bit	Semantics
0	<code>src1 == src2 && !unord(src1, src2)</code>
1	<code>src1 != src2 && !unord(src1, src2)</code>
2	<code>src1 < src2 && !unord(src1, src2)</code>
3	<code>src1 <= src2 && !unord(src1, src2)</code>
4	<code>src1 > src2 && !unord(src1, src2)</code>
5	<code>src1 >= src2 && !unord(src1, src2)</code>
6	<code>!unord(src1, src2)</code>
7	<code>unord(src1, src2)</code>
8	<code>src1 == src2 unord(src1, src2)</code>
9	<code>src1 != src2 unord(src1, src2)</code>
10	<code>src1 < src2 unord(src1, src2)</code>
11	<code>src1 <= src2 unord(src1, src2)</code>
12	<code>src1 > src2 unord(src1, src2)</code>
13	<code>src1 >= src2 unord(src1, src2)</code>

4 Default Memory Map

Address	Description
0xffffffff800	Boot ROM
...	...
0xffffffffbff	Boot ROM
0xffffffff00	Interrupt enable
0xffffffff04	Interrupt mask
0xffffffff08	Interrupt pending
0xffffffff0c	Interrupt source
0xffffffff40	Interrupt vector 0
...	...
0xffffffff7c	Interrupt vector 15
0xffffffff88	UART status
0xffffffff8c	UART data
0xffffffffa0	Timer cycles, low word
0xffffffffa4	Timer cycles, high word
0xffffffffa8	Timer nanoseconds, low word
0xffffffffac	Timer nanoseconds, high word
0xffffffffb0	Timer microseconds
0xffffffffb4	Timer seconds
0xffffffffc0	Revision
0xffffffffc4	Frequency
0xffffffffc8	Features
0xffffffffcc	Address width
0xffffffffd0	SRAM address width
0xffffffffd4	DRAM address width
0xffffffffd8	Flash address width
0xffffffffdc	I/O address width
0xffffffe0	PC width/method cache size
0xffffffe4	Direct-mapped cache size
0xffffffe8	Fully associative cache size
0xfffffec	Stack cache size
0xfffffff8	Memory spill register
0xfffffffc	Halt/reset

5 Notes

- Stores to the stack are write-back, stores to other caches are write-through. Stores do not pull data into the caches (no write allocation).
- Support for floating-point operations is optional.
- Branches use a delay-slot if bit d is set, and do not use a delay slot if it is cleared
- Loading from \$mul registers adds src2 to the value.