rtfItanium

“If you build it, he will come” – Field of Dreams

Features

- 80-bit data path, double-extended floating-point
- 64 entry general purpose register file with unified floating-point and integer registers
- 3-way out-of-order (ooo) superscalar execution
- 40-bit instructions, three per 128-bit bundle
- Instruction L1, L2 and data L1, L2 caches
- 7 entry write buffer
- Dual memory channels

Nomenclature

The ISA refers to primitive objects sizes following the convention suggested by Knuth of using Greek.

Number of Bits		Instructions
8	byte	LDB, STB
16	wyde	LDW, STW
32	tetra	LDT, STT
40	penta	LDP, STP
64	octa	LDO, STO
80	deci	LDD, STD

The register used to address instructions is referred to as the instruction pointer or IP register. The instruction pointer is a synonym for program counter or PC register.

Programming Model

Registers

The ISA is a 64-register machine with registers able to hold either integer or floating-point values. There are a large number of control and status (CSR) registers which hold an assortment of specific values relevant to processing.

Register Usage Convention

The register usage convention probably has more to do with software than hardware. Excepting a couple of special cases, the registers are general purpose in nature.

R0 always has the value zero. r61 is the link register used implicitly by the call instruction.

Register	Description / Suggested Usage	Saver
r0	always reads as zero	
r1-r4	return values / exception	caller
r5-r22	temporaries	caller
r23-r37	register variables	callee
r38-r47	function arguments	caller
r48	reserved for system	
r49	reserved for system	
r50	reserved for system	
r51	reserved for system	
r52	garbage collector	
r53	garbage collector	
r54	assembler usage	
r55	type number / function argument	caller
r56	class pointer / function argument	caller
r57	thread pointer	callee
r58	global pointer	
r59	exception SP offset	
r60	exception link register	callee
r61	return address / link register	callee
r62	base / frame pointer	callee
r63	stack pointer (hardware)	callee

Exceptions

External Interrupts

There is very little difference between an externally generated exception and an internally generated one. An externally caused exception will force a BRK instruction into the instruction stream. The BRK instruction contains a cause code identifying the external interrupt source.

Polling for Interrupts

To support managed code an interrupt polling instruction (PFI) is provided in the instruction set. In some managed code environments, it is not enough to disable and enable interrupts around critical code. The code must be effectively run with interrupt disabled all the time. This makes it necessary to poll for interrupts in software. For instance, stack prologue code may cause false pointer matches for the garbage collector because stack space is allocated before the contents are defined. If the GC scan occurs on this allocated but undefined area of memory, there could be false matches.

Effect on Machine Status

The operating mode is always switched to the machine mode on exception. It's up to the machine mode code to redirect the exception to a lower operating mode when desired. Further exceptions at the same or lower interrupt level are disabled automatically. Machine mode code must enable interrupts at some point. This can be done automatically when the exception is redirected to a lower level by the REX instruction. The RTI instruction will also automatically enable further machine level exceptions.

Exception Stack

The program counter and status bits are pushed onto an internal stack when an exception occurs. This stack is only eight entries deep as that is the maximum amount of nesting that can occur. Further nesting of exceptions can be achieved by saving the state contained in the exception registers.

Exception Vectoring

Exceptions are handled through a vector table. The vector table has four entries, one for each operating level the core may be running at. The location of the vector table is determined by TVEC[0]. If the core is operating at level three for instance and an interrupt occurs vector table address number three is used for the interrupt handler. Note that the interrupt automatically switches the core to operating level zero, privilege level zero. An exception handler at the machine level may redirect exceptions to a lower level handler identified in one of the vector registers. More specific exception information is supplied in the cause register.

Operating Level	Address (If TVEC[0] contains $FFFC0000)	
0	$FFFC0000	Handler for operating level zero
1	$FFFC0020	
2	$FFFC0040	
3	$FFFC0060	

Reset

The core begins executing instructions at address $FFFFFFFFFFFFFFC0100. All registers are in an undefined state. Register set #0 is selected.

Precision

Exceptions in rtfItanium are precise. They are processed according to program order of the instructions. If an exception occurs during the execution of an instruction, then an exception field is set in the reorder buffer. The exception is processed when the instruction commits which happens in program order. If the instruction was executed in a speculative fashion, then no exception processing will be invoked unless the instruction makes it to the commit stage.

Exception Cause Codes

The following table outlines the cause code for a given purpose. These codes are specific to rtfItanium. Under the HW column an ‘x’ indicates that the exception is internally generated by the processor; the cause code is hard-wired to that use. An ‘e’ indicates an externally generated interrupt, the usage may vary depending on the system.

Cause Code		HW	Description	
1	IBE	x	instruction bus error	
2	EXF	x	Executable fault	
4	TLB	x	tlb miss	
			FMTK Scheduler	
128		e		
129	KRST	e	Keyboard reset interrupt	
130	MSI	e	Millisecond Interrupt	
131	TICK	e		
156	KBD	e	Keyboard interrupt	
157	GCS	e	Garbage collect stop	
158	GC	e	Garbage collect	
159	TSI	e	FMTK Time Slice Interrupt	
3			Control-C pressed	
20			Control-T pressed	
26			Control-Z pressed	
32	SSM	x	single step	
33	DBG	x	debug exception	
34	TGT	x	call target exception	
35	MEM	x	memory fault	
36	IADR	x	bad instruction address	
37	UNIMP	x	unimplemented instruction	
38	FLT	x	floating point exception	
39	CHK	x	bounds check exception	
40	DBZ	x	divide by zero	
41	OFL	x	overflow	
	FLT	x	floating point exception	
47				
48	ALN	x	data alignment	
49				
50	DWF	x	Data write fault	
51	DRF	x	data read fault	
52	SGB	x	segment bounds violation	
53	PRIV	x	privilege level violation	
54	CMT	x	commit timeout	
55	BD	x	branch displacement	
56	STK	x	stack fault	
57	CPF	x	code page fault	

58	DPF	x	data page fault	
60	DBE	x	data bus error	
61				
62	NMI	x	Non-maskable interrupt	
240	SYS		Call operating system (FMTK)	
241			FMTK Schedule interrupt	
255	PFI		reserved for poll-for-interrupt instruction	

DBG

A debug exception occurs if there is a match between a data or instruction address and an address in one of the debug address registers.

IADR

This exception is currently not implemented but reserved for the purpose of identifying bad instruction addresses. If the two least significant bits of the instruction address are non-zero then this exception will occur.

UNIMP

This exception occurs if an instruction is encountered that is not supported by the processor. It is not currently implemented.

OFL

If an arithmetic operation overflows (multiply, add, or shift) and the overflow exception is enabled in the arithmetic exception enable register then an OFL exception will be triggered.

FLT

A floating-point exception is triggered if an exceptional condition occurs in the floating-point unit and the exception is enabled. Please see the section on floating-point for more details.

DRF, DWF, EXF

Data read fault, data write fault, and execute fault are exceptions that are returned by the memory management unit when an attempt is made to access memory for which the corresponding access type is not allowed. For instance, if the memory page is marked as non-executable an attempt is made to load the instruction cache from the page then an execute fault EXF exception will occur.

CPF, DPF

The code page fault and data page fault exceptions are activated by the mmu if the page is not present in memory. Access may be allowed but simply unavailable.

PRIV

Some instructions and CSR registers are legal to use only at a higher operating level. If an attempt is made to use the privileged instruction by a lower operating level, then a privilege violation exception may occur. For instance, attempting to use RTI instruction from user operating level.

STK

If the value loaded into one of the stack pointer registers (the stack point sp or frame pointer fp) is outside of the bounds defined by the stack bounds registers, then a stack fault exception will be triggered.

DBE

A timeout signal is typically wired to the err_i input of the core and if the data memory does not respond with an ack_i signal fast enough an error will be triggered. This will happen most often when the core is attempting to access an unimplemented memory area for which no ack signal is generated. When the err_i input is activated during a data fetch, an exception is flagged in a result register for the instruction. The core will process the exception when the instruction commits. If the instruction does not commit (it could be a speculated load instruction) then the exception will not be processed.

IBE

A timeout signal is typically wired to the err_i input of the core and if the instruction memory does not respond with an ack_i signal fast enough and error will be triggered. This will happen most often when the core is attempting to access an unimplemented memory area for which no ack signal is generated. When the err_i input is activated during an instruction fetch, a breakpoint instruction is loaded into the cache at the address of the error.

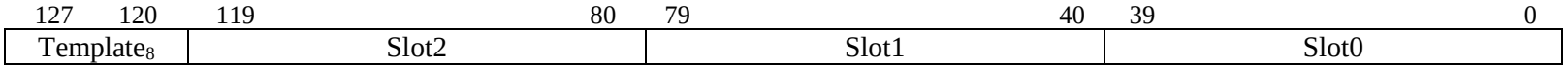
NMI

The core does not currently support non-maskable interrupts. However, this cause value is reserved for that purpose.

BD

The core will generate the BD (branch displacement) exception if a branch instruction points back to itself. Branch instructions in this sense include jump (JMP) and call (CALL) instructions.

Bundle Format:



Instruction Formats:

Immediate7			A	R	Immed15							Rs16	Opcode4	Rd6				ML
Immediate7			A	R	Immed9			Rs26				Rs16	Opcode4	Immed6				MS
Immediate7			A	R	Immed9			DC3	IC3			Rs16	Eh4	Immed6				CACHE
~7			A	R	~3	~6		Rs26				636	Dh4	636				PUSH
Immediate7			A	R	Immed15							636	Bh4	636				PUSHC
Funct5		Immed2	A	R	Sc3	Rs36		Immed6				Rs16	Opcode4	Rd6				MLX
Funct5		Immed2	A	R	Sc3	Rs36		Rs26				Rs16	Opcode4	Immed6				MSX
145		Immed2	A	R	Sc3	Rs36		DC3	IC3			Rs16	Fh4	Immed6				CACHEX
255		~2	~	~	~3	~6		~6				~6	Opcode4	~6				MEMDB
245		~2	~	~	~3	~6		~6				~6	Opcode4	~6				MEMSB
Immediate7			Op2		Immed15							Rs16	Opcode4	Rd6				RI
Funct5		02	02		~3	~6		~6				Rs16	14	Rd6				R1
Funct5		Funct22	02		Sz3	Rs36		Rs26				Rs16	Opcode4	Rd6				R3
Funct4	Rg3		o	w	~3	Rs36		Bw6				Bo6	Bh4	Rd6				BF
Funct4	Rg3		o	Bw4		Rs36		Rs26				Bo6	Bh4	Rd6				BFI
Td22..13					To12..5			Rs26				Rs16	Opcode4	To4..2	Cond3			Bcc
Td22..13					To12..5			Bitno6..1				Rs16	5h4	To4..2	B0	Cn2		BBS
Td22..13					To12..5			Immed6				Rs16	Opcode4	To4..2	Imm3			BEQI
~9					~3	Rs36		Rs26				Rs16	2h4	~2	Cond4			BRG
~9					~3	~6		~6				~6	3h4	~6				NOP
Immediate7			~2		Immed15							Rs16	8h6	616				JAL
Address30													Opcode4	Address6				CALL
Immediate18								616				636	Bh4	636				RET
H	04		~5		Cause8			~2	Imask4			Rs16	Fh4	~6				BRK
0	14		~5		2558			~2	04			06	Fh4	~6				PFI
05			~7		~6			~6				Rs16	Eh4	Sema6				RTI
15			~5		PrivLvl8			T2	Imask4			Rs16	Eh4	~6				REX

2_5	$\sim_7$	$\sim_6$	$\sim_6$	$\sim_6$	Eh_4	$\sim_6$	SYNC
3_5	$\sim_7$	$\sim_6$	$\sim_2$ $Imask_4$	$Rs1_6$	Eh_4	Rd_6	SEI
4_5	$\sim_7$	$Immed_6$	$Rs2_6$	$Rs1_6$	Eh_4	$\sim_6$	WAIT
$Immediate_7$	~ ~	$Immed_9$	$Rs2_6$	$Rs1_6$	Ch_4	$Immed_6$	CHKI
$\sim_{12}$	$Rs3_6$	$Rs2_6$	$Rs1_6$	Dh_4	$\sim_6$	CHK
$Funct_5$	$Prec_4$ Rm_3	$Rs3_6$	$Rs2_6$	$Rs1_6$	$Opcode_4$	Rd_6	FLT3
$\sim_5$	$Prec_4$ Rm_3	Op_6	$Rs2_6$	$Rs1_6$	1_4	Rd_6	FLT2
$Immediate_7$ Op_2	$Immediate_{15}$	$Rs1_6$	$Opcode_4$	Rd_6	FLTLDI
Op_2 OL_2 $\sim_3$ 0_2 $\sim_3$	$Regno_{12}$	$Rs1_6$	5_4	Rd_6	CSR
$Funct_5$ Cmd_4	$\sim_{11}$ Tn_4	$Rs1_6$	$Opcode_4$	Rd_6	TLB

Template bits 0 to 6 combined with the instruction slot number determine which functional unit the instructions is for. Template bit 7 is reserved and should be set to zero.

Memory Loads

$Opcode_4$	xx00	xx01	xx10	xx11
00xx	LDB	LDC	LDP	LDD
01xx	LDBU	LDCU	LDPU	LDDR
10xx	LDT	LDO		
11xx	LDTU	LDOU	LEA	{MLX}

Indexed Memory Loads

$Funct_5$	xx000	xx001	xx010	xx011	xx100	xx101	xx110	xx111
00xx	LDBX	LDCX	LDPX	LDDX	LDBUX	LDCUX	LDPUX	LDDRX
01xx	LDTX	LDOX			LDTUX	LDOUX	LEAX	
10xx								
11xx								

Memory Stores

$Opcode_4$	xx00	xx01	xx10	xx11
00xx	STB	STC	STP	STD
01xx				STDC
10xx	STT	STO	CAS	PUSHC
11xx	TLB	PUSH	CACHE	{MSX}

Indexed Memory Stores / Misc

$Funct_5$	xx000	xx001	xx010	xx011	xx100	xx101	xx110	xx111
00xx	STBX	STCX	STPX	STDX				STDCX
01xx	STTX	STOX	CASX				CACHE	
10xx								
11xx	MEMSB	MEMDB						

Flow Control

$Opcode_4$	xx00	xx01	xx10	xx11
00xx	Bcc	BLcc	BRcc	NOP
01xx	FBcc	BBc	BEQ #	BNE #
10xx	JAL	JMP	CALL	RET
11xx	CHK #	CHK	RTI / REX / Misc	BRK

Floating Point

Opcode$_4$ - Bits 6 to 9

Opcode$_4$	xx00	xx01	xx10	xx11
00xx		{FLT2}	FAND #	FOR #
01xx	FMA	FMS	FNMA	FNMS
10xx				
11xx				

Op$_6$ – Bits 22 to 27

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x					FADD	FSUB	FCMP		FMUL	FDIV			FAND	FOR		
1x	FMOV		FTOI	ITOF	FNEG	FABS	FSIGN	FMAN	FNABS	FCVTSD		FCVTSQ	FSTAT	FSQRT		
2x	FTX	FCX	FEX	FDX	FRM					FCVTDS						
3x							FSYNC		FSLT	FSGE	FSLE	FSGT	FSEQ	FSNE	FSUN	

Unit		
1	Branch	
2	Integer	
3	Floating Point	
4	Memory Load	
5	Memory Store	
6		
7		

Integer ALU {bits 32, 31, Opcode$_4$}

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x		{R1}	{R3}	{R3}	ADD	CSR	CMP	CMPU	AND	OR	XOR	{BF}	BLEND			MULF
1x	SLT	SGE	SLE	SGT	SLTU	SGEU	SLEU	SGTU	SEQ	SNE		{BF}				
2x	MUL	MULU	DIV	DIVU	MOD	MODU			MADF			{BF}				
3x	FXMUL				ADDS1	ADDS2	ADDS3		ANDS1	ANDS2	ANDS3	{BF}	ORS1	ORS2	ORS3	

ALU – R3 Format {Funct$_5$, bit 6}

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	MULH	MULUH	ADDV	SUBV	ADD	SUB	CMP	CMPU	AND	OR	XOR		NAND	NOR	XNOR	MULF
1x	SLT	SGE	SLE	SGT	SLTU	SGEU	SLEU	SGTU	SEQ	SNE	MOV	CMOVNZ	MIN	MAX	PTRDIF	
2x	MUL	MULU	DIV	DIVU	MOD	MODU			MADF	MUX			MAJ			
3x	FXMUL	FXDIV	SHL	ASL	SHR	ASR	ROL	ROR	SHL #	ASL #	SHR #	ASR #	ROL #	ROR #	BMM	

ALU – R1 Format Funct$_5$

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	CNTLZ	CNTLO	CNTPOP	COM	ABS	NOT	ISPTR	NEG	ZXT	ZXC	ZXB	ZXP	ZXO			
1x									SXT	SXC	SXB	SXP	SXO			

Templates

Template	Slot0	Slot1	Slot2
00	Mem Unit	Int Unit	Int Unit
01	Mem Unit	Int Unit	Int Unit
02	Mem Unit	Int Unit	Int Unit
03	Mem Unit	Int Unit	Int Unit
04	Mem Unit		
05	Mem Unit		
06			
07			
08	Mem Unit	Mem Unit	Int Unit
09	Mem Unit	Mem Unit	Int Unit
0A	Mem Unit	Mem Unit	Int Unit
0B	Mem Unit	Mem Unit	Int Unit
0C	Mem Unit	FP Unit	Int Unit
0D	Mem Unit	FP Unit	Int Unit
0E	Mem Unit	Mem Unit	FP Unit
0F	Mem Unit	Mem Unit	FP Unit
10	Mem Unit	Int Unit	Branch Unit
11	Mem Unit	Int Unit	Branch Unit
12	Mem Unit	Branch Unit	Branch Unit
13	Mem Unit	Branch Unit	Branch Unit
14			
15			
16	Branch Unit	Branch Unit	Branch Unit
17	Branch Unit	Branch Unit	Branch Unit
18	Mem Unit	Mem Unit	Branch Unit
19	Mem Unit	Mem Unit	Branch Unit
1A			
1B			
1C	Mem Unit	FP Unit	Branch Unit
1D	Mem Unit	FP Unit	Branch Unit
1E			
1F			

Template	Slot0	Slot1	Slot2
00	Int	Int	Int
01	MemLd	Int	Int
02	Int	MemLd	Int
03	MemLd	MemLd	Int
04	Int	Int	MemLd
05	MemLd	Int	MemLd
06	Int	MemLd	MemLd
07	MemLd	MemLd	MemLd
08	Branch	Int	Int
09	Int	Branch	Int
0A	Branch	Branch	Int
0B	Int	Int	Branch
0C	Branch	Int	Branch
0D	Int	Branch	Branch
0E	Branch	Branch	Branch
0F	FP	Int	Int
10	int	FP	int
11	FP	FP	int
12	Int	Int	FP
13	FP	Int	FP
14	Int	FP	FP
15	FP	FP	FP
16	Branch	MemLd	MemLd
17	MemLd	Branch	MemLd
18	Branch	Branch	MemLd
19	MemLd	MemLd	Branch
1A	Branch	MemLd	Branch
1B	MemLd	Branch	Branch
1C	Branch	FP	FP
1D	FP	Branch	FP
1E	Branch	Branch	FP
1F	FP	FP	Branch

Template	Slot0	Slot1	Slot2
20	Branch	FP	Branch
21	FP	Branch	Branch
22	MemLd	FP	FP
23	FP	MemLd	FP
24	MemLd	MemLd	FP
25	FP	FP	MemLd
26	MemLd	FP	MemLd
27	FP	MemLd	MemLd
28	MemSt	MemLd	MemLd
29	MemLd	MemSt	MemLd
2A	MemSt	MemSt	MemLd
2B	MemSt	MemLd	MemSt
2C	MemLd	MemSt	MemSt
2D	MemLd	MemLd	MemSt
2E	MemLd	MemSt	Int
2F	MemSt	MemLd	Int
30	Int	MemLd	MemSt
31	Int	MemSt	MemLd
32	MemLd	Int	MemSt
33	MemSt	Int	MemLd
34	Branch	MemLd	MemSt
35	Branch	MemSt	MemLd
36	MemLd	Branch	MemSt
37	MemSt	Branch	MemLd
38	MemLd	MemSt	Branch
39	MemSt	MemLd	Branch
3A	FP	MemLd	MemSt
3B	FP	MemSt	MemLd
3C	MemLd	FP	MemSt
3D	MemSt	FP	MemLd
3E	MemLd	MemSt	FP
3F	MemSt	MemLd	FP

Template	Slot0	Slot1	Slot2
40			
41	MemSt	Int	Int
42	Int	MemSt	Int
43	MemSt	MemSt	Int
44	Int	Int	MemSt
45	MemSt	Int	MemSt
46	Int	MemSt	MemSt
47	MemSt	MemSt	MemSt
48	MemSt	FP	FP
49	FP	MemSt	FP
4A	MemSt	MemSt	FP
4B	FP	FP	MemSt
4C	MemSt	FP	MemSt
4D	FP	MemSt	MemSt
4E	Branch	Int	MemLd
4F	Branch	Int	MemSt
50	Branch	Int	FP
51	FP	Int	Branch
52			
53			
54			
55			
56	Branch	MemSt	MemSt
57	MemSt	Branch	MemSt
58	Branch	Branch	MemSt
59	MemSt	MemSt	Branch
5A	Branch	MemSt	Branch
5B	MemSt	Branch	Branch
5C			
5D			
5E			
5F			

Template	Slot0	Slot1	Slot2
60			
61			
62			
63			
64			
65			
66			
67			
68			
69			
6A			
6B			
6C			
6D			
6E			
6F			
70			
71			
72			
73			
74			
75			
76			
77			
78			
79			
7A			
7B			
7C			
7D			
7E			
7F			

Instruction Descriptions

ABS – Absolute Value

Description:

This instruction takes the absolute value of a register and places the result in a target register.

Instruction Format: Integer R1

Clock Cycles: 0.33

Execution Units: Integer ALU

Operation:

```
If Ra < 0
   Rt = -Ra
else
   Rt = Ra
```

Exceptions: none

ADD - Addition

Description:

Add two values. The first operand must be in a register. The second operand may be in a register or may be an immediate value specified in the instruction.

Instruction Formats: Integer RI and R3

The R3 format adds three registers together.

Clock Cycles: 0.33

Execution Units: All ALU's

Exceptions: none

Notes:

ADDS1 – Addition Shifted 22 Bits

Description:

Add a register and an immediate value shifted to the left 22 bits. The immediate constant associated with the RI form of the instruction is sign extended to the left and zero extended to the right of the constant supplied in the instruction. This instruction allows building an 80-bit constant in a register when combined with other shifting instructions.

Instruction Format: Integer RI

Clock Cycles: 0.33

Execution Units: All Integer ALUs

Exceptions: none

ADDS2 – Addition Shifted 44 Bits

Description:

Add a register and an immediate value shifted to the left 44 bits. The immediate constant associated with the RI form of the instruction is sign extended to the left and zero extended to the right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other shifting instructions.

Instruction Format: Integer RI

Clock Cycles: 0.33

Execution Units: All Integer ALUs

Exceptions: none

ADDS3 – Addition Shifted 66 Bits

Description:

Add a register and an immediate value shifted to the left 66 bits. The immediate constant associated with the RI form of the instruction is zero extended to the right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other shifting instructions.

Instruction Format: Integer RI

Clock Cycles: 0.33

Execution Units: All Integer ALUs

Exceptions: none

Bcc – Conditional Branch

Description:

If the branch condition is true, the target address is computed and loaded into the program counter. The branch is relative to the address of the branch instruction. The target address uses displace-plus-offset addressing to determine the address to branch to. The branch range is approximately +/- 4MB. If the instruction branches back to itself, a branch target exception will occur.

Target Address Calculation:

The low order 13 bits of the target address are loaded directly into the instruction pointer from the To_{13} field of the instruction. The high order bits of the target are calculated as the sum of the high order instruction pointer bits and a sign extended constant found in the Td field of the instruction.

Instruction Format:

Instruction Format:

For the register form of the instruction, an absolute address is loaded into the target register if the branch condition is true.

$Cond_3$	Mne.	
0	BEQ	Rs1 = Rs2 signed
1	BNE	Rs1 <> Rs2
2	BLT / BGT	Rs1 < Rs2 or Rs2 > Rs1
3	BGE / BLE	Rs1 >= Rs2 or Rs2 <= Rs1
4	reserved	
5	reserved	
6	BLTU	Rs1 < Rs2 (unsigned)
7	BGEU	Rs1 >= Rs2 (unsigned)

Clock Cycles:

Typically, 2 with correct branch outcome and target prediction.

Execution Units: FCU Only

Exceptions: branch target

BFINSI – Bitfield Insert Immediate

Description:

A bitfield is inserted into the target register Rd by combining a value read from Rs3 with a constant shifted to the left. The bitfield may not be larger than six bits. To accommodate a larger field multiple instructions can be used, or a value loaded into a register and the BFINS instruction used.

Instruction Format: Integer BFI

Rg_2 Bit	
0	1= Bo is a register spec, 0 = Bo is a six bit immediate
1	1 = Bw is a register spec, 0 = Bw is a six bit immediate

$Rg_{[1]}$ bit should always be clear for this instruction

Clock Cycles: 1

Execution Units: ALU #0 Only

Exceptions: none

BMM – Bit Matrix Multiply

BMM Rd, Rs1, Rs2

Description:

The BMM instruction treats the bits of register Rs1 and Rs2 as an 8x8 bit matrix, performs a bit matrix multiply of the two registers and stores the result in the target register. An alternate mnemonic for this instruction is MOR. Only the least significant 64 bits of the registers are used.

Instruction Format: Integer R3

$Func_2$	Function
0	MOR
1	MXOR
2	MORT (MOR transpose)
3	MXORT (MXOR transpose)

Operation:

```
for I = 0 to 7
        for j = 0 to 7
                Rt.bit[i][j] = (Ra[i][0]&Rb[0][j]) | (Ra[i][1]&Rb[1][j]) | … | (Ra[i][7]&Rb[7][j])
```

Clock Cycles: 1

Execution Units: ALU #0 only

Exceptions: none

Notes:

The bits are numbered with bit 63 of a register representing I,j = 0,0 and bit 0 of the register representing I,j = 7,7.

BRK – Hardware / Software Breakpoint

Description:

Invoke the break handler routine. The break handler routine handles all the hardware and software exceptions in the core. A cause code is loaded into the CAUSE CSR register. The break handler should read the CAUSE code to determine what to do. The break handler is located by TVEC[0]. This address should contain a jump to the break handler. Note the reset address is $F[…]FFC0100. An exception will automatically switch the processor to the machine level operating mode. The break handler routine may redirect the exception to a lower level using the REX instruction.

The core maintains an internal eight level interrupt stack for each of the following:

Item Stacked	CSR reg	
instruction pointer	ip_stack	
operating level	ol_stack	available as a single CSR
privilege level	pl_stack	available as a single CSR
interrupt mask	im_stack	available as a single CSR

If further nesting of interrupts is required, the stacks may be copied to memory as they are available from CSR's.

On stack underflow a break exception is triggered.

Hardware interrupts will cause a BRK instruction to be inserted into the instruction stream.

Because instructions are fetched in bundles a hardware interrupt always intercepts at a bundle address, and always returns to a bundle address.

Instruction Format: BRK

H = 1 = software interrupt – return address is next instruction

H = 0 = hardware interrupt – return address is current instruction

$IMask_4$ = the priority level of the hardware interrupt, the priority level at time of interrupt is recorded in the instruction, the interrupt mask will be set to this level when the instruction commits. This field is not used for software interrupts and should be zero.

Cause Code = numeric code associated with the cause of the interrupt. The cause code is bitwise 'or'd with the value in register Rs1 to set the the cause CSR. Usually either one of the cause code field or Rs1 will be zero.

The empty instruction fields may be used to pass constant data to the break handler.

Operating Level	Address (If TVEC[0] contains $FFFC0000)	
0	$FFFC0000	Handler for operating level zero
1	$FFFC0020	
2	$FFFC0040	
3	$FFFC0060	

CACHE – Cache Command

CACHE Cmd, d[Rn]
CACHE Cmd, d[Ra + Rc * scale]

Description:

This instruction commands the cache controller to perform an operation. Commands are summarized in the command table below. Commands may be issued to both the instruction and data cache at the same time.

Instruction Formats: CACHE

Commands:

Cmd_3	Mne.	Operation
0	NOP	no operation
1	enable	enable cache (instruction cache is always enabled)
2	disable	not valid for the instruction cache
3	invline	invalidate line associated with given address
4	invall	invalidate the entire cache (address is ignored)

Operation:

Register Indirect with Displacement Form

$Line = round_{32}(sign\ extend(memory[displacement + Ra]))$

Register-Register Form

$Line = round_{32}(sign\ extend(memory[Ra + Rc * scale]))$

Notes:

Sc_2 Code	Multiply By
0	1
1	2
2	4
3	8

CALL – Call Method

Description:

This instruction loads the instruction pointer with a constant value specified in the instruction. In addition, the address of the instruction following the CALL is stored in the return address register r61. This instruction may be used to implement subroutine calls.

Instruction Format: CALL

This format has a 64GB range.

If an address range greater than 36 bits is required, then the JAL instruction must be used.

Execution Units: Branch Unit

Clock Cycles:

CHK – Check Register Against Bounds

Description:

A register is compared to two values. If the register is outside of the bounds defined by Rs2 and Rs3 or an immediate value then an exception will occur. Rs1 must be greater than or equal to Rs2 and Rs1 must be less than Rs3 or the immediate.

Instruction Format: CHK, CHKI

Clock Cycles: 0.33

Exceptions: bounds check

Notes:

The system exception handler will typically transfer processing back to a local exception handler.

JAL – Jump-And-Link

Description:

Instruction Format:

This instruction loads the program counter with the sum of a register and a constant value specified in the instruction. In addition, the address of the instruction following the JAL is stored in the specified target register. This instruction may be used to implement subroutine calls and returns.

Instruction Format: RI

Execution Units: Branch

Clock Cycles:

Exceptions: Branch target

JMP – Jump to Address

Description:

A jump is made to the address specified in the instruction.

Instruction Format: CALL

The format modifies only instruction pointer bits 0 to 35. The high order IP bits are not affected. This allows accessing code within a 64GB region of memory. Note that with the use of a mmu this address range is often sufficient.

Execution Units: Branch

Clock Cycles: 1

Exceptions: Branch target

Notes:

If an address range larger than 36 bits is required, then the value must be loaded into a register and the JAL instruction used.

The jump instruction executes immediately during the fetch stage of the core. This makes it much faster than a JAL.

LDB – Load Byte

Description:

This instruction loads a byte (8 bit) value from memory. The value is sign extended to 80 bits when placed in the target register.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDBU – Load Unsigned Byte

Description:

This instruction loads a byte (8 bit) value from memory. The value is zero extended to 80 bits when placed in the target register.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDD – Load Deci Byte

Description:

This instruction loads a byte (80 bit) value from memory

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDDR – Load Deci Byte and Reserve

Description:

This instruction loads a byte (80 bit) value from memory and places a reservation on the address range associated with the given address. The region of memory which is reserved depends on what is supported by the external memory system hardware. Typically a 128 bit region or larger would be reserved.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDO – Load Octa Byte

Description:

This instruction loads a byte (64 bit) value from memory. The value is sign extended to 80 bits when placed in the target register.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDOU – Load Unsigned Octa Byte

Description:

This instruction loads a byte (64 bit) value from memory. The value is zero extended to 80 bits when placed in the target register.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDP – Load Penta Byte

Description:

This instruction loads a penta-byte (40 bit) value from memory. The value is sign extended to 80 bits when placed in the target register.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDPU – Load Unsigned Penta Byte

Description:

This instruction loads a penta-byte (40 bit) value from memory. The value is zero extended to 80 bits when placed in the target register.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDT – Load Tetra Byte

Description:

This instruction loads a penta-byte (32 bit) value from memory. The value is sign extended to 80 bits when placed in the target register.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDTU – Load Unsigned Tetra Byte

Description:

This instruction loads a penta-byte (32 bit) value from memory. The value is zero extended to 80 bits when placed in the target register.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDW – Load Wyde

Description:

This instruction loads a wyde (16 bit) value from memory. The value is sign extended to 80 bits when placed in the target register.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

LDWU – Load Unsigned Wyde

Description:

This instruction loads a byte (16 bit) value from memory. The value is zero extended to 80 bits when placed in the target register.

Instruction Formats: ML, MLX

Clock Cycles: 4 minimum depending on memory access time

Sc_3	Scale Rb By
0	1
1	2
2	4
3	8
4	16
5	5
6	10
7	15

MAJ – Majority Logic

Description:

Determines the majority logic bits of three values in registers Rs1, Rs2, and Rs3 and places the result in the target register Rd.

Instruction Format: R3

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

Operation:

Rd = (Rs1 & Rs2) | (Rs1 & Rs3) | (Rs2 & Rs3)

OR – Bitwise Or

Description:

Perform a bitwise or operation between operands. The immediate constant associated with the RI form of the instruction is zero extended to the size of the register.

Instruction Format: Integer RI and R3

Clock Cycles: 0.33

Execution Units: All Integer ALUs

Exceptions: none

ORS1 – Bitwise Or Shifted 22 Bits

Description:

Perform a bitwise or operation between operands. The immediate constant associated with the RI form of the instruction is zero extended to the left and right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other ‘or’ and ‘or’ shifting instructions.

Instruction Format: Integer RI

Clock Cycles: 0.33

Execution Units: All Integer ALUs

Exceptions: none

ORS2 – Bitwise Or Shifted 44 Bits

Description:

Perform a bitwise or operation between operands. The immediate constant associated with the RI form of the instruction is zero extended to the left and right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other ‘or’ and ‘or’ shifting instructions.

Instruction Format: Integer RI

Clock Cycles: 0.33

Execution Units: All Integer ALUs

Exceptions: none

ORS3 – Bitwise Or Shifted 66 Bits

Description:

Perform a bitwise or operation between operands. The immediate constant associated with the RI form of the instruction is zero extended to the right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other ‘or’ and ‘or’ shifting instructions.

Instruction Format: Integer RI

Clock Cycles: 0.33

Execution Units: All Integer ALUs

Exceptions: none

PFI – Poll for Interrupt

Description:

The poll for interrupt instruction polls the interrupt status lines and performs an interrupt service if an interrupt is present. Otherwise the PFI instruction is treated as a NOP operation. Polling for interrupts is performed by managed code. PFI provides a means to process interrupts at specific points in running software.

Note that the core records the address including the slot number of the PFI instruction as the exceptioned address. This may be in the middle of an instruction bundle. In the case where an exception is present instructions in the bundle following the PFI instruction are not executed. After completing the exception service routine the core will return to the instruction following the PFI instruction, this may be in the middle of an instruction bundle.

Instruction Format: PFI

Clock Cycles: 0.33 (if no exception is present)

Execution Units: Branch

Exceptions: any outstanding hardware interrupts

PTRDIF – Difference Between Pointers

Description:

Subtract two values then shift the result right. Both operands must be in a register. The top 20 bits of each value are masked off before the subtract. The right shift is provided to accommodate common object sizes. It may still be necessary to perform a divide operation after the PTRDIF to obtain an index into odd sized or large objects.

Instruction Format: R3

Operation:

Rt = (Ra – Rb) >> Sc

Clock Cycles: 0.33

Execution Units: Integer

Exceptions:

None.

PUSH – Push Word (80 bits)

Description:

This instruction decrements the stack pointer and stores a word (80 bit) value to stack memory. The value pushed onto the stack may come from either a register or a constant value defined in the instruction.

Instruction Format: PUSH, PUSHC

Operation:

$Memory_8[SP - 10] = Rs2$
$SP = SP – 10$

Clock Cycles: 4 minimum, depending on memory access time

Execution Units: Memory Store

Notes:

Stores always write through to memory and therefore take a significant number of clock cycles before they are ready to be committed. Exceptions are checked for during the execution of a store operation.

The instruction explicitly encodes the stack pointer register r63, if desired a different register may be chosen. However, stack bounds checking is available with register r63.

SEI – Set Interrupt Mask

SEI #3
SEI $v0,#7

Description:

The interrupt level mask is set to the value specified by the instruction. The value used is the bitwise or of the contents of register Rs1 and an immediate (M_4) supplied in the instruction. The assembler assumes a mask value of fifteen, masking all interrupts, if no mask value is specified. Usually either M_4 or Rs1 should be zero. The previous setting of the interrupt mask is stored in Rd.

Instruction Format: SEI

Operation:

Rd = im
im = M_4 | Rs1