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 object 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 instruction pointer 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. The last two registers r62 and r63 are used for stack references and subject to stack bounds checking.

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

Branches

Target Address

Branches use absolute target addressing. The branch target field of the instruction is large enough that relative addressing is not required. Branching modifies the low order 23 bits of the instruction pointer while leaving the remaining bits as they are. Branching then takes place within an 8MB page of memory.

Performance

Branches should target the first slot (slot0) of a bundle for maximum performance. The core always fetches instructions at bundle aligned addresses. Branching into the middle of a bundle lowers performance because effectively fewer instructions are available for queueing. Branches should also be placed in the last slot (slot2) for maximum performance.

Shift Operations

The immediate constant for a shift is limited to six bits or the values 0 to 63. If it's required to shift by more than 63 bits two shift operations will be needed, or the value of the shift may come from a register.

Stores

The write buffer is dual ported and will accept two store operations at the same time. They will be placed in the write queue in program order.

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 instruction pointer 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 $FFFFFFFFFFFFFFFFC0100. 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:

127 120	119 80	79 40	39 0
Template$_8$	Slot2	Slot1	Slot0

Instruction Formats:

Immediate$_7$				A	R	Immed$_{15}$				Rs1$_6$	Opcode$_4$	Rd$_6$			ML
Immediate$_7$				A	R	Immed$_9$		Rs2$_6$		Rs1$_6$	Opcode$_4$	Immed$_6$			MS
Immediate$_7$				A	R	Immed$_9$		DC$_3$	IC$_3$	Rs1$_6$	Eh$_4$	Immed$_6$			CACHE
~$_7$				A	R	~$_3$	~$_6$	Rs2$_6$		63$_6$	Dh$_4$	63$_6$			PUSH
Immediate$_7$				A	R	Immed$_{15}$				63$_6$	Bh$_4$	63$_6$			PUSHC
Funct$_5$			Immed$_2$	A	R	Sc$_3$	Rs3$_6$	Immed$_6$		Rs1$_6$	Opcode$_4$	Rd$_6$			MLX
Funct$_5$			Immed$_2$	A	R	Sc$_3$	Rs3$_6$	Rs2$_6$		Rs1$_6$	Opcode$_4$	Immed$_6$			MSX
Funct$_4$	0		~$_2$	A	R	Sz$_3$	~$_6$	Rs2$_6$		Rs1$_6$	Bh$_4$	Rd$_6$			AMO
Funct$_4$	1		Immed$_2$	A	R	Sz$_3$	Immed$_{12}$			Rs1$_6$	Bh$_4$	Rd$_6$			AMOI
14$_5$			Immed$_2$	A	R	Sc$_3$	Rs3$_6$	DC$_3$	IC$_3$	Rs1$_6$	Fh$_4$	Immed$_6$			CACHEX
25$_5$			~$_2$	~	~	~$_3$	~$_6$	~$_6$		~$_6$	Opcode$_4$	~$_6$			MEMDB
24$_5$			~$_2$	~	~	~$_3$	~$_6$	~$_6$		~$_6$	Opcode$_4$	~$_6$			MEMSB
Immediate$_7$				Op$_2$		Immed$_{15}$				Rs1$_6$	Opcode$_4$	Rd$_6$			RI
Funct$_5$			0$_2$	0$_2$		~$_3$	~$_6$	~$_6$		Rs1$_6$	1$_4$	Rd$_6$			R1
Funct$_5$			Funct2$_2$	0$_2$		Sz$_3$	Rs3$_6$	Rs2$_6$		Rs1$_6$	Opcode$_4$	Rd$_6$			R3
Funct$_5$			Funct2$_2$	0$_2$		Sz$_3$	~$_6$	Rs2$_6$		Rs1$_6$	Opcode$_4$	Rd$_6$			SHIFT
Funct$_5$			Funct2$_2$	0$_2$		Sz$_3$	~$_6$	Immed$_6$		Rs1$_6$	Opcode$_4$	Rd$_6$			SHIFTI
Funct$_4$		Rg$_3$		o	w	~$_3$	Rs3$_6$	Bw$_6$		Bo$_6$	Bh$_4$	Rd$_6$			BF
Funct$_4$		Rg$_3$		o	Bw$_4$		Rs3$_6$	Rs2$_6$		Bo$_6$	Bh$_4$	Rd$_6$			BFINS
Address$_{22..5}$								Rs2$_6$		Rs1$_6$	Opcode$_4$	Ad$_{4..2}$	Cond$_3$		Bcc
Address$_{22..5}$								Bitno$_{6..1}$		Rs1$_6$	5h$_4$	Ad$_{4..2}$	B$_0$	Cn$_2$	BBS
Address$_{22..5}$								Immed$_6$		Rs1$_6$	Opcode$_4$	Ad$_{4..2}$	Imm$_3$		BEQI
~$_9$						~$_3$	Rs3$_6$	Rs2$_6$		Rs1$_6$	2h$_4$	~$_2$	Cond$_4$		BRG
~$_9$						~$_3$	~$_6$	~$_6$		~$_6$	3h$_4$	~$_6$			NOP
Immediate$_7$				~$_2$		Immed$_{15}$				Rs1$_6$	8h$_6$	61$_6$			JAL
Address$_{37..8}$											Opcode$_4$	Address$_{7..2}$			CALL
Immediate$_{19..1}$								61$_6$		63$_6$	Bh$_4$	63$_6$			RET

<table>
<tr><td>H</td><td colspan="3">0_4</td><td colspan="4">$\sim_5$</td><td colspan="2">$Cause_8$</td><td>$\sim_2$</td><td>$Imask_4$</td><td>$Rs1_6$</td><td>Fh_4</td><td>$\sim_6$</td><td>BRK</td></tr>
<tr><td>0</td><td colspan="3">1_4</td><td colspan="4">$\sim_5$</td><td colspan="2">255_8</td><td>$\sim_2$</td><td>0_4</td><td>0_6</td><td>Fh_4</td><td>$\sim_6$</td><td>PFI</td></tr>
<tr><td colspan="4">0_5</td><td colspan="5">$\sim_7$</td><td>$\sim_6$</td><td colspan="2">$\sim_6$</td><td>$Rs1_6$</td><td>Eh_4</td><td>$Sema_6$</td><td>RTI</td></tr>
<tr><td colspan="4">1_5</td><td colspan="4">$\sim_5$</td><td colspan="2">$PrivLvl_8$</td><td>T_2</td><td>$Imask_4$</td><td>$Rs1_6$</td><td>Eh_4</td><td>$\sim_6$</td><td>REX</td></tr>
<tr><td colspan="4">2_5</td><td colspan="5">$\sim_7$</td><td>$\sim_6$</td><td colspan="2">$\sim_6$</td><td>$\sim_6$</td><td>Eh_4</td><td>$\sim_6$</td><td>SYNC</td></tr>
<tr><td colspan="4">3_5</td><td colspan="5">$\sim_7$</td><td>$\sim_6$</td><td>$\sim_2$</td><td>$Imask_4$</td><td>$Rs1_6$</td><td>Eh_4</td><td>Rd_6</td><td>SEI</td></tr>
<tr><td colspan="4">4_5</td><td colspan="5">$\sim_7$</td><td>$Immed_6$</td><td colspan="2">$Rs2_6$</td><td>$Rs1_6$</td><td>Eh_4</td><td>$\sim_6$</td><td>WAIT</td></tr>
<tr><td colspan="5">$Immediate_7$</td><td>~</td><td>~</td><td colspan="3">$Immed_9$</td><td colspan="2">$Rs2_6$</td><td>$Rs1_6$</td><td>Ch_4</td><td>$Immed_6$</td><td>CHKI</td></tr>
<tr><td colspan="9">$\sim_{12}$</td><td>$Rs3_6$</td><td colspan="2">$Rs2_6$</td><td>$Rs1_6$</td><td>Dh_4</td><td>$\sim_6$</td><td>CHK</td></tr>
<tr><td colspan="4">$Funct_5$</td><td colspan="3">$Prec_4$</td><td colspan="2">Rm_3</td><td>$Rs3_6$</td><td colspan="2">$Rs2_6$</td><td>$Rs1_6$</td><td>$Opcode_4$</td><td>Rd_6</td><td>FLT3</td></tr>
<tr><td colspan="4">$\sim_5$</td><td colspan="3">$Prec_4$</td><td colspan="2">Rm_3</td><td>Op_6</td><td colspan="2">$Rs2_6$</td><td>$Rs1_6$</td><td>1_4</td><td>Rd_6</td><td>FLT2</td></tr>
<tr><td colspan="5">$Immediate_7$</td><td colspan="2">Op_2</td><td colspan="5">$Immediate_{15}$</td><td>$Rs1_6$</td><td>$Opcode_4$</td><td>Rd_6</td><td>FLTLDI</td></tr>
<tr><td colspan="2">Op_2</td><td>OL_2</td><td colspan="2">$\sim_3$</td><td colspan="2">0_2</td><td colspan="2">$\sim_3$</td><td colspan="3">$Regno_{12}$</td><td>$Rs1_6$</td><td>5_4</td><td>Rd_6</td><td>CSR</td></tr>
<tr><td colspan="4">$Funct_5$</td><td colspan="3">Cmd_4</td><td colspan="4">$\sim_{11}$</td><td>Tn_4</td><td>$Rs1_6$</td><td>$Opcode_4$</td><td>Rd_6</td><td>TLB</td></tr>
</table>

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	LDW	LDP	LDD
01xx	LDBU	LDWU	LDPU	LDDR
10xx	LDT	LDO		{AMO}
11xx	LDTU	LDOU	LEA	{MLX}

Indexed Memory Loads

Funct$_5$	xx000	xx001	xx010	xx011	xx100	xx101	xx110	xx111
00xx	LDBX	LDWX	LDPX	LDDX	LDBUX	LDWUX	LDPUX	LDDRX
01xx	LDTX	LDOX			LDTUX	LDOUX	LEAX	
10xx								
11xx								

AMO

Funct$_5$	xx000	xx001	xx010	xx011	xx100	xx101	xx110	xx111
00xx	AMOSWAP	AMOSWAPI	AMOADD	AMOADDI	AMOAND	AMOANDI	AMOOR	AMORI
01xx	AMOXOR	AMOXORI	AMOSHL	AMOSHLI	AMOSHR	AMOSHRI	AMOMIN	AMOMINI
10xx	AMOMAX	AMOMAXI	AMOMINU	AMOMINUI	AMOMAXU	AMOMAXUI		
11xx								

Memory Stores

$Opcode_4$	xx00	xx01	xx10	xx11
00xx	STB	STW	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	STWX	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	Branch	MemLd	Int
53	MemLd	Branch	Int
54	Int	Branch	MemLd
55	Int	MemLd	Branch
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	Branch	MemSt	Int
5D	Int	Branch	MemSt
5E	MemSt	Branch	Int
5F	MemSt	Int	Branch

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

Instruction Descriptions

Execution Units: this is the functional unit that the instruction is executed by.

Clock cycles:

This is a highly optimistic estimate of the number of clock cycles required to complete the operation. This estimate assumes all values required for the operation are available without delay. Under ideal circumstances many operations can complete in 1/3 of a clock cycle because there are enough functional units to service that many instructions at once.

Preload Instructions

Issuing a load instruction with a target register of r0 causes the cache line to be loaded without updating the register file. If an exception occurs during the load, it will be ignored.

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

AMO – Atomic Memory Operation

Description:

The atomic memory operations read from memory addressed by the Rs1 register and store the value in Rd. As a second step the value from memory is combined with the value in register Rs2 or an immediate constant according to one of the available functions then stored back into the memory addressed by Rs1.

Instruction Format: AMO, AMOI

$Funct_4$	Mnemonic	Operation Performed	
00	amoswap	swap	memory[Ra] = Rb
01	amoadd	addition	memory[Ra] = memory[Ra] + Rb
02	amoand	bitwise and	memory[Ra] = memory[Ra] & Rb
03	amoor	bitwise or	memory[Ra] = memory[Ra] | Rb
04	amoxor	bitwise exclusive or	memory[Ra] = memory[Ra] ^ Rb
05	amoshl	shift left	memory[Ra] = memory[Ra] << Rb
06	amoshr	shift right	memory[Ra] = memory[Ra] >> Rb
07	amomin	minimum	memory[Ra] = memory[Ra] < Rb ? memory[Ra] : Rb
08	amomax	maximum	memory[Ra] = memory[Ra] >Rb ? memory[Ra] : Rb
09	amominu	minimum unsigned	memory[Ra] = memory[Ra] < Rb ? memory[Ra] : Rb
0A	amomaxu	maximum unsigned	memory[Ra] = memory[Ra] > Rb ? memory[Ra] : Rb

Sz_3	
0	Byte
1	Wyde
2	Tetra
3	Penta
4	Octa
5	Deci
6	reserved
7	reserved

Acquire and release bits determine the ordering of memory operations.

A = acquire – 1 = no following memory operations can take place before this one

R = release – 1 = this memory operation cannot take place before prior ones.

All combinations of A, R are allowed.

AND – Bitwise And

Description:

Perform a bitwise 'and' operation between operands. For the immediate form, the immediate constant is one extended to the left before use.

Instruction Format: RI, R3

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

ASL – Arithmetic Shift Left

Description:

Bits from the source register Rs1 are shifted left by the amount in register Rs2 or an immediate value. A zero is shifted into bit zero. The difference between this instruction and a SHL instruction is that ASL may cause an arithmetic overflow exception. SHL will never cause an exception.

The immediate constant for a shift is limited to six bits or the values 0 to 63. If it's required to shift by more than 63 bits two shift operations will be needed, or the value of the shift may come from a register.

Instruction Format: SHIFT, SHIFTI

Clock Cycles: 0.33

Execution Units: Integer

Exceptions:

An overflow exception may result if the bits shifted out from the MSB are not the same as the resulting sign bit and the exception is enabled in the AEC register. Exceptions are only caused by a word size operation.

ASR – Arithmetic Shift Right

Description:

Bits from the source register Rs1 are shifted right by the amount in register Rs2 or an immediate value. The sign bit is shifted into the most significant bits preserving the sign of the value.

The immediate constant for a shift is limited to six bits or the values 0 to 63. If it's required to shift by more than 63 bits two shift operations will be needed, or the value of the shift may come from a register.

Instruction Formats: SHIFT, SHIFTI

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

BAND –Branch on Logical And

Description:

If the logical and of two registers is true, a target address is loaded into the instruction pointer. Only bits 0 to 22 of the instruction pointer are affected. If the branch branches back to itself a branch exception will be generated. The branch range is approximately 8MB.

Instruction Format: Bcc, BRcc

Operation:

```
if (!(Ra && Rb))
        ip[22:0] = target
```

Clock Cycles:

Under absolutely ideal conditions, 0.33 clocks are required. Ideal conditions being the branch is located in instruction slot #2 (so that the prior two slots may be executing instructions). Also register values must already be valid and the branch prediction is correct. Branches are predicted and taken in the fetch stage of the processor.

Execution Units: Branch

Exceptions: branch target

BBC –Branch if Bit Clear

Description:

If the specified bit in a register is clear, the target address is loaded into the instruction pointer. Only the low order 23 bits of the instruction pointer are affected. If the branch branches back to itself a branch exception will be generated. The range of the branch is approximately 8MB.

Instruction Format: BBc

Operation:

```
if (Ra[bitno]=0)
        ip[22:0] = target
```

Clock Cycles: 0.33 with accurate prediction, otherwise 8 or more

Execution Units: branch

Exceptions: branch target address

BBS –Branch if Bit Set

Description:

If the specified bit in a register is set, the target address is loaded into the instruction pointer. Only the low order 23 bits of the instruction pointer are affected. If the branch branches back to itself a branch exception will be generated. The range of the branch is approximately 8MB.

Instruction Format: BBc

Operation:

```
if (Ra[bitno]=1)
    ip[22:0] = target
```

Clock Cycles: 0.33 with accurate prediction, otherwise 8 or more

Execution Units: branch

Exceptions: branch target address

Bcc – Conditional Branch

Description:

If the branch condition is true, the target address is loaded into the instruction pointer. Only bits 0 to 22 of the instruction pointer are affected. If the instruction branches back to itself, a branch target exception will occur. The range of the branch is approximately 8MB.

Instruction Format: Bcc

$Cond_3$	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:

Under absolutely ideal conditions, 0.33 clocks are required. Ideal conditions being the branch is located in instruction slot #2 (so that the prior two slots may be executing instructions). Also register values must already be valid and the branch prediction is correct. Branches are predicted and taken in the fetch stage of the processor.

Execution Units: Branch

Exceptions: branch target

BEQ –Branch if Equal

Description:

If the two registers are equal, a target address is loaded into the instruction pointer. Only bits 0 to 22 of the instruction pointer are affected. If the branch branches back to itself a branch exception will be generated. The branch range is approximately 8MB.

Instruction Format: Bcc, BRcc

Operation:

```
if (Rs1 = Rs2)
        ip[22:0] = target
```

Clock Cycles:

Under absolutely ideal conditions, 0.33 clocks are required. Ideal conditions being the branch is located in instruction slot #2 (so that the prior two slots may be executing instructions). Also register values must already be valid and the branch prediction is correct. Branches are predicted and taken in the fetch stage of the processor.

Execution Units: Branch

Exceptions: branch target

BEQI –Branch if Equal Immediate

Description:

If a register is equal to nine-bit sign extended value, a target address is loaded into the instruction pointer. Only bits 0 to 22 of the instruction pointer are affected. If the branch branches back to itself a branch exception will be generated. The branch range is approximately 8MB. This instruction is particularly useful for implementing case statements based on small values.

Instruction Format: Bcc

There is no register target form for this instruction.

Operation:

if (Ra = Imm_9)
    ip[22:0] = target

Clock Cycles:

Under absolutely ideal conditions, 0.33 clocks are required. Ideal conditions being the branch is located in instruction slot #2 (so that the prior two slots may be executing instructions). Also register values must already be valid and the branch prediction is correct. Branches are predicted and taken in the fetch stage of the processor.

Execution Units: Branch

Exceptions: branch target

BFCHG – Bitfield Change

Description:

A bitfield in the source specified by Da is inverted, the result is copied to the target register. Bo specifies the bit offset. Bw specifies the bit width. The bit width and offset may either be contained in a register or an immediate value.

Instruction Format: BF

Rg_3 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
2	1

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

BFCLR – Bitfield Clear

Description:

A bitfield is cleared in the target register. All bits are copied from a source Da (which is zero extended if an immediate value) except for bits identified by the bitfield which are set to zero in the target.

Instruction Format: BF

Rg_3 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
2	1 = Da is a register spec, 0 = Da is an sixteen bit immediate

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

Notes:

Normally Da is a register which is the same as the target register Rt.

BFEXT – Bitfield Extract

Description:

A bitfield is extracted from the source register Da by shifting to the right and ‘and’ masking. The result is sign extended to the width of the machine. This instruction may be used to sign extend a value from an arbitrary bit position.

Instruction Format: BF

Rg_3 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
2	1 = Da is a register spec, 0 = Da is an sixteen bit immediate

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

Notes:

While it is possible for Da to be a constant the instruction would not normally be used this way.

BFEXTU – Bitfield Extract

Description:

A bitfield is extracted from the source register Da by shifting to the right and 'and' masking. The result is zero extended to the width of the machine. This instruction may be used to zero extend a value from an arbitrary bit position.

Instruction Format: BF

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

BFFFO – Bitfield Find First One

Description:

A bitfield contained in Da is searched beginning at the most significant bit to the least significant bit for a bit that is set. The index into the word of the bit that is set is stored in Rd. If no bits are set, then Rd is set equal to -1. To get the index into the bitfield of the set bit, subtract off the bitfield offset.

Instruction Format: BF

Rg_3 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
2	1 = Da is a register spec, 0 = Da is an sixteen bit immediate

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

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

BGE –Branch if Greater or Equal

Description:

If the value in Rs1 is greater than or equal to the value in Rs2, a target address is loaded into the instruction pointer. The values are treated as signed numbers. Only bits 0 to 22 of the instruction pointer are affected. If the branch branches back to itself a branch exception will be generated. The branch range is approximately 8MB.

Instruction Format: Bcc, BRcc

Operation:

```
if (Rs1 >= Rs2)
    ip[22:0] = target
```

Clock Cycles:

Under absolutely ideal conditions, 0.33 clocks are required. Ideal conditions being the branch is located in instruction slot #2 (so that the prior two slots may be executing instructions). Also register values must already be valid and the branch prediction is correct. Branches are predicted and taken in the fetch stage of the processor.

Execution Units: Branch

Exceptions: branch target

BGEU –Branch if Greater or Equal

Description:

If the value in Rs1 is greater than or equal to the value in Rs2, a target address is loaded into the instruction pointer. The values are treated as unsigned numbers. Only bits 0 to 22 of the instruction pointer are affected. If the branch branches back to itself a branch exception will be generated. The branch range is approximately 8MB.

Instruction Format: Bcc, BRcc

Operation:

```
if (Rs1 >= Rs2)
    ip[22:0] = target
```

Clock Cycles:

Under absolutely ideal conditions, 0.33 clocks are required. Ideal conditions being the branch is located in instruction slot #2 (so that the prior two slots may be executing instructions). Also register values must already be valid and the branch prediction is correct. Branches are predicted and taken in the fetch stage of the processor.

Execution Units: Branch

Exceptions: branch target

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.

BRcc – Conditional Branch to Register

Description:

This instruction allows the target address to be supplied by a register, accommodating computed target addresses. If the branch condition is true, the target address is loaded into the instruction pointer. All bits of the instruction pointer are affected. If the instruction branches back to itself, a branch target exception will occur.

Instruction Format: BRcc

Cond$_4$	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	BNAND	!(Rs1 && Rs2)
5	BNOR	!(Rs1 || Rs2)
6	BLTU	Rs1 < Rs2 (unsigned)
7	BGEU	Rs1 >= Rs2 (unsigned)
8	FBEQ	Rs1 = Rs2 floating-point compare
9	FBNE	Rs1 != Rs2
A	FBLT	Rs1 < Rs2
B	FBLE	Rs1 <= Rs2
C	BAND	Rs1 && Rs2
D	BOR	Rs1 || Rs2
E	reserved	
F	FBUN	unordered comparison

Clock Cycles:

Execution Units: Branch

Exceptions: branch target

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. The target return address register is assumed to be r61 by the instruction. In order to use a different register, the JAL instruction must be used.

Instruction Format: CALL

This format has a 256GB range. The format modifies only instruction pointer bits 0 to 37. The high order IP bits are not affected. Note that with the use of a mmu this address range is often sufficient.

If an address range greater than 38 bits is required, then the JAL instruction must be used.

Execution Units: Branch Unit

Clock Cycles: 0.33

Notes:

Ideally the call instruction is placed in slot2 and targets an address in slot0. Many routines can be aligned on a bundle address. Since there are three instructions in a bundle the average extra space consumed by aligning routines on a bundle address is only 1.5 instruction words.

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

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.

CNTLO – Count Leading Ones

Description:

Count the number of leading ones (starting at the MSB) in Rs1 and place the count in the target register.

Instruction Format: R1

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

CNTLZ – Count Leading Zeros

Description:

Count the number of leading zeros (starting at the MSB) in Rs1 and place the count in the target register.

Instruction Format: R1

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

CNTPOP – Count Population

Description:

Count the number of one bits in Rs1 and place the count in the target register.

Instruction Format: R1

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

CSR – Control and Status Access

Description:

The CSR instruction group provides access to control and status registers in the core. For the read-write operation the current value of the CSR is placed in the target register Rd then the CSR is updated from register Rs1. The CSR read / update operation is an atomic operation.

Instruction Format: CSR

Op_2		Operation
0	CSRRD	Only read the CSR, no update takes place, Rs1 should be R0.
1	CSRRW	Both read and write the CSR
2	CSRRS	Read CSR then set CSR bits
3	CSRRC	Read CSR then clear CSR bits

CSRRS and CSRRC operations are only valid on registers that support the capability.

The OL_2 field is reserved to specify the operating level. Note that registers cannot be accessed by a lower operating level.

$Regno_{12}$		Access	Description
001	HARTID	R	hardware thread identifier (core number)
002	TICK	R	tick count, counts every cycle from reset
030-037	TVEC	RW	trap vector handler address
048	EPC	RW	exceptioned pc, pc value at point of exception
044	STATUSL	RWSC	status register, contains interrupt mask, operating level
045	STATUSH	RW	status register bits 64 to 127
080-0BF	CODE	RW	code buffers
FF0	INFO	R	Manufacturer name
FF1	“	R	“
FF2	“	R	cpu class
FF3	“	R	“
FF4	“	R	cpu name
FF5	“	R	“
FF6	“	R	model number
FF7	“	R	serial number
FF8	“	R	cache sizes instruction (bits 40 to 79), data (bits 0 to 39)

Execution Units: Integer, the instruction may be available on only a single execution unit (not supported on all available integer units).

Clock Cycles: 0.33

Exceptions: privilege violation attempting to access registers outside of those allowed for the operating level.

JAL – Jump-And-Link

Description:

Instruction Format:

This instruction loads the instruction pointer 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: 4

This instruction may not take effect until it is executed during the execute stage of the processor. It will cause the pipeline to flush which will dump following instructions from the queue. Depending on the size of the queue an average of ½ the number of entries in the queue will be dumped.

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 37. The high order IP bits are not affected. This allows accessing code within a 256GB region of memory. Note that with the use of a mmu this address range is often sufficient.

Execution Units: Branch

Clock Cycles: 0.33

Exceptions: Branch target

Notes:

If an address range larger than 38 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.

JMP should be placed in slot2 for best performance.

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 (80 bits)

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 (64 bits)

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 (40 bits)

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 (32 bits)

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 (16 bits)

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)

MUX – Multiplex

Description:

The MUX instruction performs a bit-by-bit copy of a bit of Rs2 to the target register Rd if the corresponding bit in Rs1 is set, or a copy of a bit from Rs3 if the corresponding bit in Ra is clear.

Instruction Format: R3

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

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: Integer

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: Integer

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:

$\text{Memory}_8[\text{SP - 10}] = \text{Rs2}$
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.

RET – Return from Subroutine

Description:

This instruction performs a subroutine return by loading the instruction pointer with the contents of the return address register (r61). Additionally, the stack pointer is adjusted by a constant supplied in the instruction. The immediate constant is a multiple of two to keep the stack word aligned. The constant is also zero extended to the left.

Instruction Format: RET

PC = RA
SP = SP + Immediate * 2

Clock Cycles: 1 (more if predicted incorrectly).

Execution Units: Branch

Exceptions: none

Notes:

The registers used to implement the stack and return address are specified in the instruction format so different registers may be used. However, future versions of the core may not support this feature.

The RET instruction is detected and used at the fetch stage of the processor to update the RSB.

REX – Redirect Exception

Description:

This instruction redirects an exception from an operating level to a lower operating level and privilege level. If the target operating level is hypervisor then the hypervisor privilege level (1) is set. If the target operating level is supervisor, then one of the supervisor privilege levels must be chosen (2 to 6). This instruction if successful jumps to the target exception handler and does not return. If this instruction fails execution will continue with the next instruction.

This instruction may fail if exceptions are not enabled at the target level.

When redirecting the target privilege level is set to the bitwise 'or' of an immediate constant specified in the instruction and register Rs1. One of these two values should be zero. The result should be a value in the range 2 to 255. The instruction will not allow setting the privilege level numerically less than the operating level.

The location of the target exception handler is found in the trap vector register for that operating level (tvec[xx]).

The cause (cause) and bad address (badaddr) registers of the originating level are copied to the corresponding registers in the target level.

The REX instruction also specifies the interrupt mask level to set for further processing.

Attempting to redirect the operating level to the machine level (0) will be ignored. The instruction will be treated as a NOP with the exception of setting the interrupt mask register.

Instruction Format: REX

Tgt_2	
0	not used
1	redirect to hypervisor level
2	redirect to supervisor level
3	not used

Clock Cycles: 4

Execution Units: Branch

Example:

```
REX 5,12,r0      ; redirect to supervisor handler, privilege level two
; If the redirection failed, exceptions were likely disabled at the target level.
; Continue processing so the target level may complete it's operation.
RTI              ; redirection failed (exceptions disabled ?)
```

Notes:

Since all exceptions are initially handled at the machine level the machine level handler must check for disabled lower level exceptions.

RTI – Return from Interrupt

Description:

Return from an interrupt or exception processing subroutine. The interrupted instruction pointer is loaded into the instruction pointer register. The internal interrupt stack is popped and the operating level, privilege level, interrupt mask level, and register set are reset to values before the exception occurred. Optionally a semaphore bit in the semaphore register is cleared. The least significant bit of the semaphore register (the reservation status bit) is always cleared by this instruction.

Instruction Format: RTI

Semaphore[$Sema_6$|[Rs1]] = 0
Semaphore[0] = 0

Clock Cycles: 8 minimum

Execution Units: Branch

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

STB – Store Byte (8 bits)

Description:

This instruction stores a byte (8 bit) value to memory.

Instruction Format: MS, MSX

Operation:

Memory_8[Rs1 + immediate] = Rs2

Clock Cycles: 4 minimum depending on memory access time

The number of clock cycles required by this instruction may be “hidden” because the core will continue to process instructions while the store is being processed.

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

Execution Units: Memory Store

Exceptions: DBE, TLB, WRV

Notes:

A store instruction will only be issued by the core if it is known that it will be able to commit without being interrupted by a change of program flow due to a previous instruction. 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.

STD – Store Deci (80 bits)

Description:

This instruction stores a deci byte (80 bit) value to memory.

Instruction Format: MS, MSX

Operation:

$\text{Memory}_{80}[\text{Rs1} + \text{immediate}] = \text{Rs2}$

Clock Cycles: 4 minimum depending on memory access time

The number of clock cycles required by this instruction may be “hidden” because the core will continue to process instructions while the store is being processed.

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

Execution Units: Memory Store

Notes:

A store instruction will only be issued by the core if it is known that it will be able to commit without being interrupted by a change of program flow due to a previous instruction. 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.

STO – Store Octa (64 bits)

Description:

This instruction stores an octa byte (64 bit) value to memory.

Instruction Format: MS, MSX

Operation:

$\text{Memory}_{64}[\text{Rs1} + \text{immediate}] = \text{Rs2}$

Clock Cycles: 4 minimum depending on memory access time

The number of clock cycles required by this instruction may be “hidden” because the core will continue to process instructions while the store is being processed.

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

Execution Units: Memory Store

Notes:

A store instruction will only be issued by the core if it is known that it will be able to commit without being interrupted by a change of program flow due to a previous instruction. 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.

STP – Store Penta (40 bits)

Description:

This instruction stores a penta (40 bit) value to memory. This operation is ½ the 80-bit word size of the machine, and also the size of an instruction in an instruction bundle.

Instruction Format: MS, MSX

Operation:

$\text{Memory}_{40}[\text{Rs1} + \text{immediate}] = \text{Rs2}$

Clock Cycles: 4 minimum depending on memory access time

The number of clock cycles required by this instruction may be "hidden" because the core will continue to process instructions while the store is being processed.

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

Execution Units: Memory Store

Notes:

A store instruction will only be issued by the core if it is known that it will be able to commit without being interrupted by a change of program flow due to a previous instruction. 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.

STT – Store Tetra (32 bits)

Description:

This instruction stores a tetra (32 bit) value to memory.

Instruction Format: MS, MSX

Operation:

$\text{Memory}_{32}[\text{Rs1} + \text{immediate}] = \text{Rs2}$

Clock Cycles: 4 minimum depending on memory access time

The number of clock cycles required by this instruction may be “hidden” because the core will continue to process instructions while the store is being processed.

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

Execution Units: Memory Store

Notes:

A store instruction will only be issued by the core if it is known that it will be able to commit without being interrupted by a change of program flow due to a previous instruction. 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.

STW – Store Wyde (16 bits)

Description:

This instruction stores a wyde (16 bit) value to memory.

Instruction Format: MS, MSX

Operation:

$\text{Memory}_{16}[\text{Rs1} + \text{immediate}] = \text{Rs2}$

Clock Cycles: 4 minimum depending on memory access time

The number of clock cycles required by this instruction may be “hidden” because the core will continue to process instructions while the store is being processed.

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

Execution Units: Memory Store

Notes:

A store instruction will only be issued by the core if it is known that it will be able to commit without being interrupted by a change of program flow due to a previous instruction. 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.

TLB – TLB Command

Description:

The command is executed on the TLB unit. The command results are placed in internal TLB registers which can be read or written using TLB command instruction. If the operation is a read register operation, then the register value is placed into Rd. If the operation is a write register operation, then the value for the register comes from Rs1. Otherwise the Rs1/Rd field in the instruction is ignored.

This instruction is only available at the machine operating level.

Instruction Format: TLB

Clock Cycles: 3

It typically takes several cycles for the TLB to process the command.

Tn_4 – This field identifies which TLB register is being read or written.

Tn_4		Assembler
0	Wired	Wired
1	Index	Index
2	Random	Random
3	Page Size	PageSize
4	Virtual page	VirtPage
5	Physical page	PhysPage
7	ASID	ASID
8	Miss address	MA
9	reserved	
10	Page Table Address	PTA
11	Page Table Control	PTC
12	Aging frequency control	AFC

TLB Commands

Cmd_4	Description	Assembler
0	No operation	
1	Probe TLB entry	TLBPB
2	Read TLB entry	TLBRD
3	Write TLB entry corresponding to random register	TLBWR
4	Write TLB entry corresponding to index register	TLBWI
5	Enable TLB	TLBEN
6	Disable TLB	TLBDIS
7	Read register	TLBRDREG
8	Write register	TLBWRREG
9	Invalidate all entries	TLBINV
10	Get age count	TLBRDAGE
11	Set age count	TLBWRAGE

Probe TLB – The TLB will be tested to see if an address translation is present.

Read TLB – The TLB entry specified in the index register will be copied to TLB holding registers.

Write Random TLB – A random TLB entry will be written into from the TLB holding registers.

Write Indexed TLB – The TLB entry specified by the index register will be written from the TLB holding registers.

Disable TLB – TLB address translation is disabled so that the physical address will match the supplied virtual address.

Enable TLB – TLB address translation is enabled. Virtual address will be translated to physical addresses using the TLB lookup tables.

The TLB will automatically update the miss address registers when a TLB miss occurs only if the registers are zero to begin with. System software must reset the registers to zero after a miss is processed. This mechanism ensures the first miss that occurs is the one that is recorded by the TLB.

PageTableAddr – This is a scratchpad register available for use to store the address of the page table.

PageTableCtrl – This is a scratchpad register available for use to store control information associated with the page table.

Notes:

The TLB automatically tracks reference counts and ages address mappings.

ZXB – Zero Extend Byte

Description:

Zero extend a byte (8 bits) as an unsigned value to the full width of the machine.

Instruction Format: R1

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

ZXO – Zero Extend Octa Byte

Description:

Zero extend an octa byte (64 bits) as an unsigned value to the full width of the machine.

Instruction Format: R1

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

ZXP – Zero Extend Penta Byte

Description:

Zero extend a penta byte (40 bits) as an unsigned value to the full width of the machine.

Instruction Format: R1

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

ZXT – Zero Extend Tetra

Description:

Zero extend a tetra byte (32 bits) as an unsigned value to the full width of the machine.

Instruction Format: R1

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none

ZXW – Zero Extend Wyde

Description:

Zero extend a wyde byte (16 bits) as an unsigned value to the full width of the machine.

Instruction Format: R1

Clock Cycles: 0.33

Execution Units: Integer

Exceptions: none