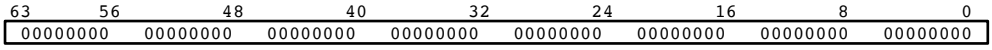
Blitz-64: ISA Quick Reference

Primary Data Type: 64 bit Signed Integer

		Bits	Bytes	Hex Example	Decimal Range
.b	"byte"	8	1	0x1f	-128 … +127
.h	"halfword"	16	2	0x1234	-32,768 … +32,767
.w	"word"	32	4	0x12345678	-2,147,483,648 … + 2,147,483,647
.d	**"doubleword"**	**64**	**8**	**0x0123_4567_89ab_cdef**	**-9,223,372,036,854,775,808 … +9,223,372,036,854,775,807**

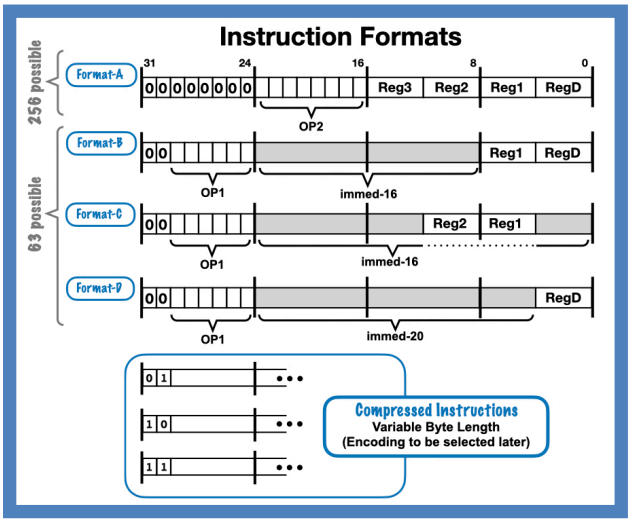


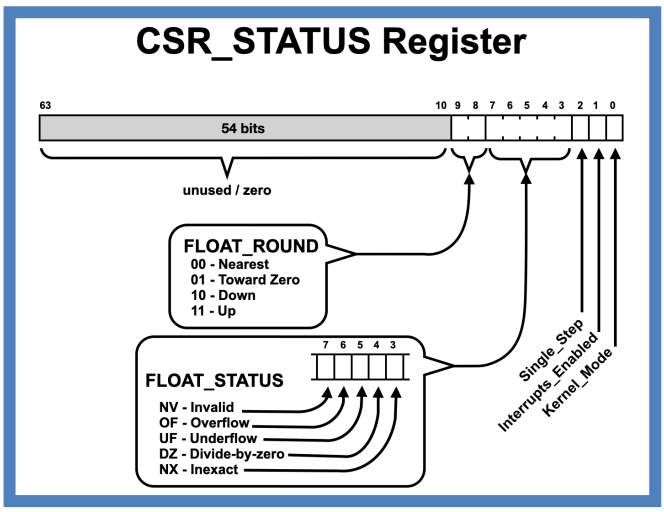
General Purpose Registers (64 bits)

r0		Zero (fixed)
r1		Argument 1 / Return Value
r2		Argument 2
r3		Argument 3
r4		Argument 4
r5		Argument 5
r6		Argument 6
r7		Argument 7
r8	t	Temp reg, used by assembler/linker
r9	s0	Work reg (caller-saved)
r10	s1	Work reg (caller-saved)
r11	s2	Work reg (caller-saved)
r12	tp	Thread data pointer
r13	gp	Global data pointer
r14	lr	Link register
r15	sp	Stack pointer

CSR: Control and Status Registers (64 bits)

0	r/o	csr_version	Version of the BLITZ-64 architecture ISA
1	r/o	csr_prod	Product Identifier
2	r/o	csr_core	Core number
3	r/o	csr_instr	Instruction counter (Reset upon power-on-reset)
4	r/o	csr_cycle	Cycle counter (Reset upon power-on-reset)
5	r/w	csr_timer	Time of next interrupt, in cycles
6	**r/w**	**csr_status**	**System Status Register**
7	r/w	csr_stat2	Previous System Status Register
8	r/w	csr_trapvec	Pointer to trap handler code
9	r/w	csr_pgtable	Pointer to page table root node
10	r/w	csr_prevpc	Previous PC (i.e., return address)
11	r/w	csr_cause	Trap code, indicating which trap just happened
12	r/w	csr_bad	Offending instruction (for Kernel Exception: Cause)
13	r/w	csr_addr	Offending Virtual Address
14	r/w	csr_ptr	Ptr to Process Control Block (& reg save area)
15	r/w	csr_temp	Temp work reg





Exceptions and Interrupts

Hardware Phase:

csr_prevpc ← PC
PC ← csr_trapvec
csr_stat2 ← csr_status
csr_status [INTERRUPTS_ENABLED] ← 0
csr_status [KERNEL_MODE] ← 1
csr_status [SINGLE_STEP] ← 0
csr_cause ← <Trap Code>
csr_bad ← <additional trap info; e.g., the offending instruction>
csr_addr ← <additional trap info; e.g., virtual address>

Global Trap Handler: Will dispatch through jump table
Trap Code: Serves as index into a software "Jump Table"
Jump Table: Contains JUMPs to Individual Handler Routines
Size of a JUMP instruction: 8 bytes max.
Masking of Asynchronous Interrupts: Single bit only (in csr_status)
Exceptions if Interrupts Disabled: Promoted to a "Kernel Exception"

Trap Codes (offset into Jump Table)

decimal	hex	
0	0000	Syscall 0
…	…	…
8184	1FF8	Syscall 1023
8192	2000	Arithmetic Exception
8200	2008	Unaligned LOAD/STORE
8208	2010	Null Address Exception
8216	2018	Illegal Instruction, Privilege Violation
8224	2020	Page Illegal Address Exception
8232	2028	Page Table Exception
8240	2030	Page Invalid Exception
8248	2038	Page Write Exception
8256	2040	Page Fetch Exception
8264	2048	Page Copy-On-Write Exception
8272	2050	Page First Dirty Exception
8280	2058	Debug Exception
8288	2060	Breakpoint Exception
8296	2068	Singlestep Exception
8304	2070	Kernel Exception
8312	2078	Emulated Instruction Exception
8320	2080	Hardware Fault Exception
8328	2088	Bad Array Index Exception
8336	2090	**Asynch:** Timer Interrupt
8344	2098	**Asynch:** DMA-Complete Interrupt

…Interrupts for other asynchronous I/O devices…

Blitz-64: ISA Quick Reference

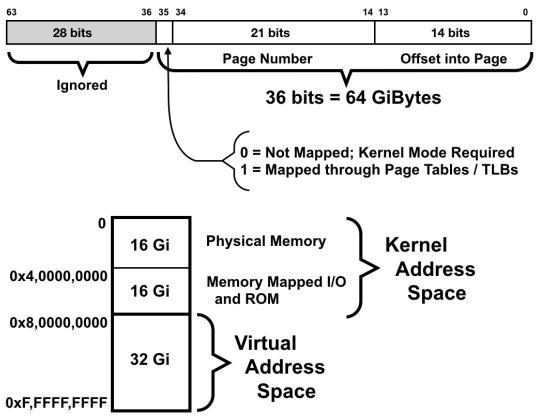
Memory

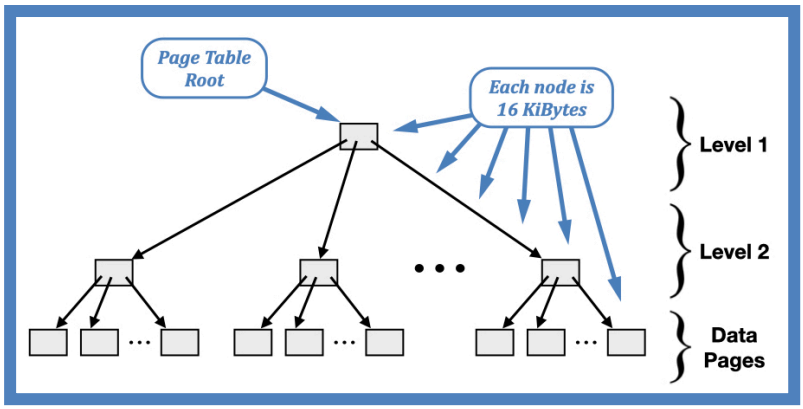
- Byte Addressable; Big Endian
- Address Size: 36 bits
- Max Virtual Address Space: 32 GB
- Max Kernel Address Space: 16 GB
- Memory Mapped I/O Region: 16 GB
- Max Physical Memory: 16 TB

Virtual Memory / Page Tables

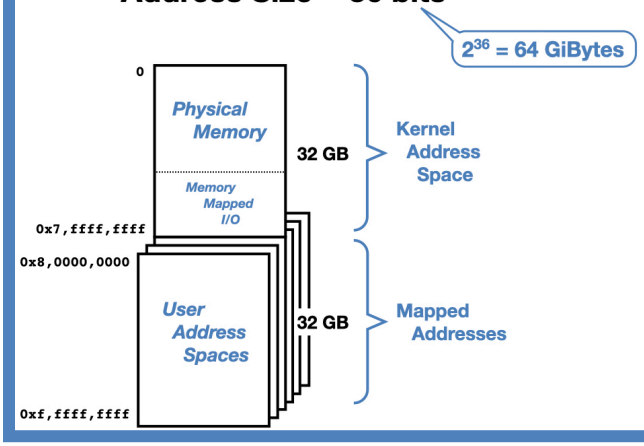
- Page Size: 16 KB
- Two-Level Table is sufficient
- Page Table Entries: 64 bits
- I/O devices can be mapped into virtual address spaces
- Address Space IDs: 16 bits

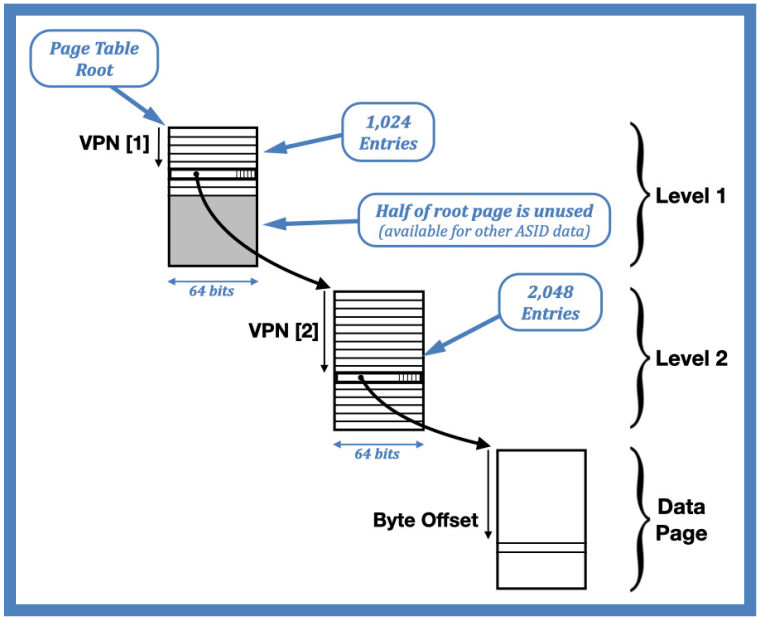
Program Generated Addresses



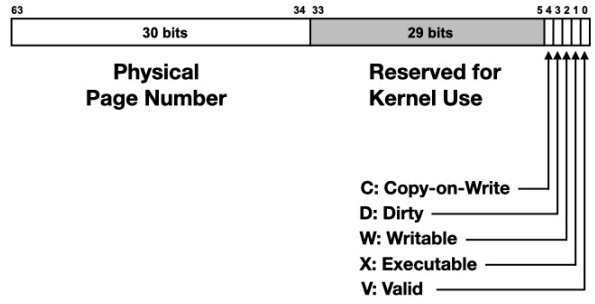


Address Size = 36 bits

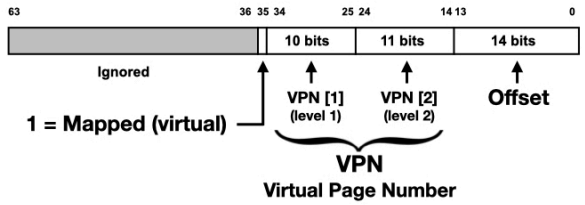




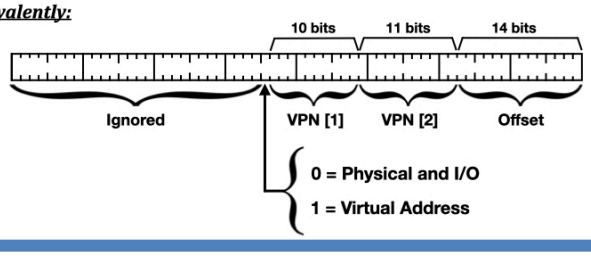
Page Table Entry (PTE):



Virtual Addresses:



Equivalently:



Blitz-64: ISA Quick Reference

	Synthetic	OP1	OP2	Exceptions	Operands	Description
Integer Arithmetic (64 bit Signed)						
	ADD		01	Arithmetic	RegD,Reg1,Reg2	RegD ← Reg1 + Reg2
	ADDI	01		Arithmetic	RegD,Reg1,immed16	RegD ← Reg1 + immed
	ADDOK		02		RegD,Reg1,Reg2	RegD ← (Reg1 + Reg2 overflows) ? 0 : 1
	ADD3		03		RegD,Reg1,Reg2,Reg3	RegD ← Reg1 + Reg2 + Reg3 (overflow ignored)
	SUB		04	Arithmetic	RegD,Reg1,Reg2	RegD ← Reg1 - Reg2
✓	NEG			Arithmetic	RegD,Reg2	RegD ← - Reg1
✓	ABS			Arithmetic	RegD,Reg2	RegD ← AbsoluteValue (Reg1)
✓	MUL			Arithmetic	RegD,Reg1,Reg2	RegD ← Reg1 × Reg2
	MULADD		05	Arithmetic	RegD,Reg1,Reg2	RegD ← (Reg1 × Reg2) + Reg3
	MULADDU		06		RegD,Reg1,Reg2,Reg3	RegD ← (Reg1 × Reg2) + Reg3 (overflow ignored)
Divide — Optional; may cause Emulated Instruction Exception						
	DIV		07	Arithmetic	RegD,Reg1,Reg2	RegD ← Reg1 ÷ Reg2
	REM		08	Arithmetic	RegD,Reg1,Reg2	RegD ← Reg1 % Reg2
Logical						
	AND		09		RegD,Reg1,Reg2	RegD ← Reg1 & Reg2
	ANDI	02			RegD,Reg1,immed16	RegD ← Reg1 & immed
	OR		0a		RegD,Reg1,Reg2	RegD ← Reg1 | Reg2
	ORI	03			RegD,Reg1,immed16	RegD ← Reg1 | immed
	XOR		0b		RegD,Reg1,Reg2	RegD ← Reg1 ^ Reg2
	XORI	04			RegD,Reg1,immed16	RegD ← Reg1 ^ immed
✓	BITNOT				RegD,Reg1	RedD ← Bitwise NOT (Reg1)
✓	LOGNOT				RegD,Reg1	RegD ← (Reg1 = 0) ? 1 : 0
Move						
✓	MOV				RegD,Reg1	RegD ← Reg1
✓	MOVI				RegD,immed64	RegD ← immed
Shift						
	SLL		0c	Arithmetic	RegD,Reg1,Reg2	RegD ← Reg1 << Reg2, Reg2 must be 0..63
	SLA		0d	Arithmetic	RegD,Reg1,Reg2	RegD ← Reg1 <<< Reg2, Reg2 must be 0..63
	SRL		0e	Arithmetic	RegD,Reg1,Reg2	RegD ← Reg1 >> Reg2, Reg2 must be 0..63
	SRA		0f	Arithmetic	RegD,Reg1,Reg2	RegD ← Reg1 >>> Reg2, Reg2 must be 0..63
	ROTR		10		RegD,Reg1,Reg2	Rotate right; Rotate left if Reg2 < 0
	SLLI	05			RegD,Reg1,immed6	RegD ← Reg1 << immed
	SLAI	06		Arithmetic	RegD,Reg1,immed6	RegD ← Reg1 <<< immed
	SRLI	07			RegD,Reg1,immed6	RegD ← Reg1 >> immed
	SRAI	08			RegD,Reg1,immed6	RegD ← Reg1 >>> immed
	ROTRI	09			RegD,Reg1,immed6	Rotate left if <0
Sign Extension						
	SEXTB		11		RegD,Reg1	Sign extend byte to 64 bits
	SEXTH		12		RegD,Reg1	Sign extend 16 bits to 64 bits
	SEXTW		13		RegD,Reg1	Sign extend 32 bits to 64 bits
Range Checking						
	NULLTEST		14	Null Address	Reg1	Trap if Reg1 is null
	CHECKB		15	Arithmetic	Reg1	Trap if Reg1 not within -128 … +127
	CHECKH		16	Arithmetic	Reg1	Trap if Reg1 not within -32768 … +32767
	CHECKW		17	Arithmetic	Reg1	Trap if Reg1 not within 32 bit range
	INDEX0		18	Bad Array Index	RegD,Reg1,Reg2,Reg3	Reg1=arrayPtr, Reg2=header, Reg3=index
	INDEX1		19	Bad Array Index	RegD,Reg1,Reg2,Reg3	. RegD ← Reg1 + 8 + (Reg3 × scale)
	INDEX2		1a	Bad Array Index	RegD,Reg1,Reg2,Reg3	. Reg2 = header = [ArrayMAX || ArrayCURR]
	INDEX4		1b	Bad Array Index	RegD,Reg1,Reg2,Reg3	. Trap if (Reg3 < 0) or (Reg3 ≥ ArrayCURR)
	INDEX8		1c	Bad Array Index	RegD,Reg1,Reg2,Reg3	. or (ArrayMAX = 0)
	INDEX16		1d	Bad Array Index	RegD,Reg1,Reg2,Reg3	.
	INDEX24		1e	Bad Array Index	RegD,Reg1,Reg2,Reg3	.
	INDEX32		1f	Bad Array Index	RegD,Reg1,Reg2,Reg3	.

Blitz-64: ISA Quick Reference

	Synthetic	OP1	OP2	Exceptions	Operands	Description
Byte Reordering						
	ENDIANH		20		RegD,Reg1	Reorder bytes in all 4 halfwords
	ENDIANW		21		RegD,Reg1	Reorder bytes in both words
	ENDIAND		22		RegD,Reg1	Reorder bytes in a doubleword
Test and Set a Boolean						
	TESTEQ		23		RegD,Reg1,Reg2	RegD ← (Reg1 = Reg2) ? 1 : 0
	TESTNE		24		RegD,Reg1,Reg2	RegD ← (Reg1 ≠ Reg2) ? 1 : 0
	TESTLT		25		RegD,Reg1,Reg2	RegD ← (Reg1 < Reg2) ? 1 : 0
	TESTLE		26		RegD,Reg1,Reg2	RegD ← (Reg1 ≤ Reg2) ? 1 : 0
✓	TESTGT				RegD,Reg1,Reg2	RegD ← (Reg1 > Reg2) ? 1 : 0
✓	TESTGE				RegD,Reg1,Reg2	RegD ← (Reg1 ≥ Reg2) ? 1 : 0
	TESTEQI	0a			RegD,Reg1,immed16	RegD ← (Reg1 = immed) ? 1 : 0
	TESTNEI	0b			RegD,Reg1,immed16	RegD ← (Reg1 ≠ immed) ? 1 : 0
	TESTLTI	0c			RegD,Reg1,immed16	RegD ← (Reg1 < immed) ? 1 : 0
	TESTLEI	0d			RegD,Reg1,immed16	RegD ← (Reg1 ≤ immed) ? 1 : 0
	TESTGTI	0e			RegD,Reg1,immed16	RegD ← (Reg1 > immed) ? 1 : 0
	TESTGEI	0f			RegD,Reg1,immed16	RegD ← (Reg1 ≥ immed) ? 1 : 0
✓	TESTEQZ				RegD,Reg1	RegD ← (Reg1 = 0) ? 1 : 0, i.e., if zero
✓	TESTNEZ				RegD,Reg1	RegD ← (Reg1 ≠ 0) ? 1 : 0, i.e., if non-zero
✓	TESTLTZ				RegD,Reg1	RegD ← (Reg1 < 0) ? 1 : 0, i.e., if negative
✓	TESTLEZ				RegD,Reg1	RegD ← (Reg1 ≤ 0) ? 1 : 0, i.e., if non-positive
✓	TESTGTZ				RegD,Reg1	RegD ← (Reg1 > 0) ? 1 : 0, i.e., if positive
✓	TESTGEZ				RegD,Reg1	RegD ← (Reg1 ≥ 0) ? 1 : 0, i.e., if non-negative
Branch - Limited Range						
	B.EQ	10		Null Address	Reg1,Reg2,immed16	Branch if Reg1 = Reg2; Offset is PC-relative
	B.NE	11		Null Address	Reg1,Reg2,immed16	Branch if Reg1 ≠ Reg2; Offset is PC-relative
	B.LT	12		Null Address	Reg1,Reg2,immed16	Branch if Reg1 < Reg2; Offset is PC-relative
	B.LE	13		Null Address	Reg1,Reg2,immed16	Branch if Reg1 ≤ Reg2; Offset is PC-relative
Branch - Unlimited Range						
✓	BEQ			Null Address	Reg1,Reg2,address	Branch if Reg1 = Reg2
✓	BNE			Null Address	Reg1,Reg2,address	Branch if Reg1 ≠ Reg2
✓	BLT			Null Address	Reg1,Reg2,address	Branch if Reg1 < Reg2
✓	BLE			Null Address	Reg1,Reg2,address	Branch if Reg1 ≤ Reg2
✓	BGT			Null Address	Reg1,Reg2,address	Branch if Reg1 > Reg2
✓	BGE			Null Address	Reg1,Reg2,address	Branch if Reg1 ≥ Reg2
✓	BEQI			Null Address	Reg,immed64,address	Branch if Reg = immediate value
✓	BNEI			Null Address	Reg,immed64,address	Branch if Reg ≠ immediate value
✓	BLTI			Null Address	Reg,immed64,address	Branch if Reg < immediate value
✓	BLEI			Null Address	Reg,immed64,address	Branch if Reg ≤ immediate value
✓	BGTI			Null Address	Reg,immed64,address	Branch if Reg > immediate value
✓	BGEI			Null Address	Reg,immed64,address	Branch if Reg ≥ immediate value
✓	BEQZ			Null Address	Reg,address	Branch if Reg = 0
✓	BNEZ			Null Address	Reg,address	Branch if Reg ≠ 0
✓	BLTZ			Null Address	Reg,address	Branch if Reg < 0, i.e., if negative
✓	BLEZ			Null Address	Reg,address	Branch if Reg ≤ 0, i.e., if not positive
✓	BGTZ			Null Address	Reg,address	Branch if Reg > 0, i.e., if positive
✓	BGEZ			Null Address	Reg,address	Branch if Reg ≥ 0, i.e., if not negative
✓	BFALSE			Null Address	Reg,address	Branch if Reg = 0, i.e., if “false”
✓	BTRUE			Null Address	Reg,address	Branch if Reg ≠ 0, i.e., if “true”

Blitz-64: ISA Quick Reference

Synthetic	OP1	OP2	Exceptions	Operands	Description
Supporting Larger Values					
UPPER20	14			RegD,immed20	RegD ← immed << 16
UPPER16	15			RegD,Reg1,immed16	RegD ← (immed << 16) + Reg1
SHIFT16	16			RegD,Reg1,immed16	RegD ← (Reg1 + immed) << 16
ADDPC	17			RegD,immed20	RegD ← immed + PC
AUIPC	18			RegD,immed20	RegD ← (immed << 16) + PC
Jumping - Limited Range					
JAL	19		Null Address	RegD,immed20	RegD ← return addr; Target ← PC+offset
JALR	1a		Null Address	RegD,immed16 (Reg1)	RegD ← return addr; Target ← offset+Reg1
Call / Jump / Return - Unlimited Range					
✓ CALL			Null Address	address	Jump to address; save return addr in “lr”
✓ CALLR			Null Address	Reg1	Jump to address; save return addr in “lr”
✓ RET			Null Address		Return value is in link reg “lr”
✓ JUMP			Null Address	address	Jump to address
✓ JR			Null Address	Reg1	Indirect jump, via register
Load - Limited Range					
LOAD.B	1b		Page, Null	RegD,immed16 (Reg1)	Sign extend 8 bits to 64 bits
LOAD.H	1c		Page, Null, Unaligned	RegD,immed16 (Reg1)	Sign extend 16 bits to 64 bits
LOAD.W	1d		Page, Null, Unaligned	RegD,immed16 (Reg1)	Sign extend 32 bits to 64 bits
LOAD.D	1e		Page, Null, Unaligned	RegD,immed16 (Reg1)	
Load - Unlimited Range					
✓ LOADB			Page, Null	RegD,address	
✓ LOADH			Page, Null, Unaligned	RegD,address	
✓ LOADW			Page, Null, Unaligned	RegD,address	
✓ LOADD			Page, Null, Unaligned	RegD,address	
✓ LOADB			Page, Null	RegD,offset (Reg1)	
✓ LOADH			Page, Null, Unaligned	RegD,offset (Reg1)	
✓ LOADW			Page, Null, Unaligned	RegD,offset (Reg1)	
✓ LOADD			Page, Null, Unaligned	RegD,offset (Reg1)	
Store - Limited Range					
STORE.B	1f		Page, Null	immed16 (Reg1),Reg2	Ignore upper 56 bits
STORE.H	20		Page, Null, Unaligned	immed16 (Reg1),Reg2	Ignore upper 48 bits
STORE.W	21		Page, Null, Unaligned	immed16 (Reg1),Reg2	Ignore upper 32 bits
STORE.D	22		Page, Null, Unaligned	immed16 (Reg1),Reg2	
Store - Unlimited Range					
✓ STOREB			Page, Null	address,Reg2	
✓ STOREH			Page, Null, Unaligned	address,Reg2	
✓ STOREW			Page, Null, Unaligned	address,Reg2	
✓ STORED			Page, Null, Unaligned	address,Reg2	
✓ STOREB			Page, Null	offset (Reg1),Reg2	
✓ STOREH			Page, Null, Unaligned	offset (Reg1),Reg2	
✓ STOREW			Page, Null, Unaligned	offset (Reg1),Reg2	
✓ STORED			Page, Null, Unaligned	offset (Reg1),Reg2	
Support for Unaligned Loads and Stores					
ALIGNH		27		RegD,Reg1,Reg2,Reg3	
ALIGNW		28		RegD,Reg1,Reg2,Reg3	
ALIGND		29		RegD,Reg1,Reg2,Reg3	
INJECT1H		2a		RegD,Reg1,Reg2,Reg3	
INJECT2H		2b		RegD,Reg1,Reg2,Reg3	
INJECT1W		2c		RegD,Reg1,Reg2,Reg3	
INJECT2W		2d		RegD,Reg1,Reg2,Reg3	
INJECT1D		2e		RegD,Reg1,Reg2,Reg3	
INJECT2D		2f		RegD,Reg1,Reg2,Reg3	

Blitz-64: ISA Quick Reference

	Synthetic	OP1	OP2	Exceptions	Operands	Description
Miscellaneous						
	SYSCALL	23		Syscall	immed10	
	SYSRET		30	Privileged		
✓	NOP					
	ILLEGAL		00	Illegal Instruction		
	SLEEP1		31	Privileged		Enter light sleep state
	SLEEP2		32	Privileged		Enter deep sleep state
	RESTART		33	Privileged		Same as power-on-reset
	DEBUG		34	Debug		
	BREAKPOINT		35	Breakpoint		
	CONTROL	24		Privileged	RegD,Reg1,immed16	Undefined; Implementor's choice
	CONTROLU	25			RegD,Reg1,immed16	Undefined; Implementor's choice
	CAS		36	Page, Null Address	RegD,Reg1,Reg2,Reg3	Compare and Set: If *r1=r2 then *r1←r3, rd←1 else rd←0
	FENCE		37			Memory barrier
CSR Manipulation						
	CSRSWAP		38	Privileged	RegD,CSRReg1,Reg2	RegD ← CSR; CSR ← Reg2
	CSRREAD		39	Privileged	RegD,CSRReg1	RegD ← CSR
✓	CSRWRITE			Privileged	CSRReg1,Reg2	CSR ← Reg2
	CSRSET	26		Privileged	CSRReg1,immed16	Set selected bits in CSR
	CSRCLR	27		Privileged	CSRReg1,immed16	Clear selected bits in CSR
	GETSTAT		3a		RegD	RegD ← csr_status & 0x0000_0000_0000_03f8
	PUTSTAT		3b		Reg1	csr_status [9:3] ← Reg1 [9:3]
Memory Management Unit						
	TLBCLEAR		3c	Privileged		Invalidate all TLBs for current ASID
	TLBFLUSH		3d	Privileged	Reg1	Invalidate TLB for the virtual address in Reg1
	CHECKADDR	28		Privileged	RegD,Reg1,immed3	Reg1 = virt addr; RegD←except. code or 0; immed=access type
Double Precision Floating Point — Optional: may cause Emulated Instruction Exception						
	FADD		3e		RegD,Reg1,Reg2	RegD ← Reg1 + Reg2
	FSUB		3f		RegD,Reg1,Reg2	RegD ← Reg1 - Reg2
	FMUL		40		RegD,Reg1,Reg2	RegD ← Reg1 × Reg2
	FDIV		41		RegD,Reg1,Reg2	RegD ← Reg1 / Reg2
	FMIN		42		RegD,Reg1,Reg2	RegD ← Minimum (Reg1, Reg2)
	FMAX		43		RegD,Reg1,Reg2	RegD ← Maximum (Reg1, Reg2)
	FNEG		44		RegD,Reg1	RegD ← -Reg1
	FABS		45		RegD,Reg1	RegD ← AbsoluteValue (Reg1)
	FSQRT		46		RegD,Reg1	RegD ← SquareRoot (Reg1)
	FEQ		47		RegD,Reg1,Reg2	RegD ← (Reg1 = Reg2) ? 1 : 0 (float compare)
	FLT		48		RegD,Reg1,Reg2	RegD ← (Reg1 < Reg2) ? 1 : 0 (float compare)
	FLE		49		RegD,Reg1,Reg2	RegD ← (Reg1 ≤ Reg2) ? 1 : 0 (float compare)
✓	FGT				RegD,Reg1,Reg2	RegD ← (Reg1 > Reg2) ? 1 : 0 (float compare)
✓	FGE				RegD,Reg1,Reg2	RegD ← (Reg1 ≥ Reg2) ? 1 : 0 (float compare)
	FCVTFI		4a		RegD,Reg1	Convert: floating-point ← int
	FCVTIF		4b		RegD,Reg1	Convert: int ← floating-point
	FMADD		4c		RegD,Reg1,Reg2,Reg3	RegD ← (Reg1 × Reg2) + Reg3
	FNMADD		4d		RegD,Reg1,Reg2,Reg3	RegD ← (-(Reg1 × Reg2)) + Reg3
	FMSUB		4e		RegD,Reg1,Reg2,Reg3	RegD ← (Reg1 × Reg2) - Reg3
	FNMSUB		4f		RegD,Reg1,Reg2,Reg3	RegD ← (-(Reg1 × Reg2)) - Reg3