

ISEM Reference Card (Integer Unit)

• Notation

Notation	Meaning
<i>reg</i>	integer register (%r0-%r31); subscript denotes instruction field (e.g., <i>reg_{rd}</i>)
<i>const</i>	unsigned integer value; subscript denotes size in bits (e.g., <i>const₂₂</i>)
<i>disp</i>	signed integer displacement; subscript denotes size in bits (e.g., <i>disp₂₂</i>)
<i>siconst</i>	2's complement signed integer value; subscript denotes size in bits (e.g., <i>siconst₃₀</i>)
<i>address</i>	<i>reg_{rs1}</i> <i>reg_{rs1} + reg_{rs2}</i> <i>reg_{rs1} + siconst₁₃</i> <i>reg_{rs1} - siconst₁₃</i> <i>siconst₁₃</i> <i>siconst₁₃ + reg_{rs1}</i>
<i>regaddr</i>	<i>reg_{rs1}</i> <i>reg_{rs1} + reg_{rs2}</i>
<i>label</i>	an instruction label
<i>asi</i>	address space identifier (0 to 255)

• Registers

Name	Meaning
%r0-%r31	integer registers
%g0-%g7	global regs (%r0-%r7)
%o0-%o7	output regs (%r8-%r15)
%l0-%l7	local regs (%r16-%r23)
%i0-%i7	input regs (%r24-%r31)
%fp	frame pointer (%i6)
%sp	stack pointer (%o6)
%asr1-%asr31	ancillary state registers
%psr	processor status register
%tbr	trap base register
%y	multiply/divide register

%r0 always holds the 32-bit integer value 0.

%psr has the following structure:

31:28	27:24	23:22	21:20	19:14	13	12	11:8	7	6	5	4:0		
impl	ver	n	z	v	c	reserved	EC	EF	PIL	S	PS	ET	CWP

%tbr has the following structure:

31:12	11:4	3:0
Trap base address	trap type	0000

• Store Instructions

Assembler Syntax

operation *reg_{rd}*, [*address*]
operationa *reg_{rd}*, [*regaddr*]*asi*

Instruction Formats

31:30	29:25	24:19	18:14	13	12:5	4:0
11	rd	op3	rs1	0	asi	rs2
11	rd	op3	rs1	1	siconst ₁₃	

Operations

operation	op3	operation (alt. space)	op3
stb	000101	stba	010101
sth	000110	stha	010110
st	000100	sta	010100
std	000111	stda	010111

• Load Instructions

Assembler Syntax

operation [*address*], *reg_{rd}*
operationa [*regaddr*]*asi*, *reg_{rd}*

Instruction Formats

31:30	29:25	24:19	18:14	13	12:5	4:0
11	rd	op3	rs1	0	asi	rs2
11	rd	op3	rs1	1	siconst ₁₃	

Operations

operation	op3	operation (alt. space)	op3
ldsb	001001	ldsba	011001
ldsh	001010	ldsha	011010
ldub	000001	lduba	010001
lduh	000010	lduha	010010
ld	000000	lda	010000
ldd	000011	ldda	010011
ldstub	001101	ldstuba	011101
swap	001111	swapa	011111

• SETHI Instructions

Assembler Syntax

sethi *const₂₂*, *reg_{rd}*
sethi %hi(*const*), *reg_{rd}*

Instruction Format

31:30	29:25	24:22	21:0
00	rd	100	const ₂₂

• Arithmetic and Logical Instructions

Assembler Syntax

operation *reg_{rs1}*, *reg_{rs2}*, *reg_{rd}*
operation *reg_{rs1}*, *siconst₁₃*, *reg_{rd}*

Instruction Formats

31:30	29:25	24:19	18:14	13	12:5	4:0
10	rd	op3	rs1	0	unused (zero)	rs2
10	rd	op3	rs1	1	siconst ₁₃	

Operations

operation	op3	operation	op3
add	000000	addcc	010000
addx	001000	addxcc	011000
and	000001	andcc	010001
andn	000101	andncc	010101
		mulccc	100100
or	000010	orcc	010010
orn	000110	orncc	010110
udiv	001110	udivcc	011110
umul	001010	umulcc	011010
sdiv	001111	sdivcc	011111
smul	001011	smulcc	011011
sub	000100	subcc	010100
subx	001100	subxcc	011100
taddcc	100000	taddcctv	100010
tsubcc	100001	tsubcctv	100011
xor	000011	xorcc	010011
xnor	000111	xnorecc	010111

• Shift Instructions

Assembler Syntax

operation *reg_{rs1}*, *reg_{rs2}*, *reg_{rd}*
operation *reg_{rs1}*, *const₅*, *reg_{rd}*

Instruction Formats

31:30	29:25	24:19	18:14	13	12:5	4:0
10	rd	op3	rs1	0	unused (zero)	rs2
10	rd	op3	rs1	1	unused (zero)	const ₅

Operations

operation	op3
sll	100101
sra	100111
srl	100110

• Branch Instructions

Assembler Syntax

operation *label*
operation,a *label*

Instruction Format

31:30	29	28:25	24:22	21:0
00	a	cond	010	disp ₂₂

Operations

operation	cond	operation	cond
bn	0000	ba	1000
be (bz)	0001	bne (bnz)	1001
ble	0010	bg	1010
bl	0011	bge	1011
bleu	0100	bgu	1100
bcs (blu)	0101	bcc (bgeu)	1101
bneg	0110	bpos	1110
bvs	0111	bvc	1111

• Jump and Link Instructions

Assembler Syntax

jmp *address*, *reg_{rd}*

Instruction Formats

31:30	29:25	24:19	18:14	13	12:5	4:0
10	rd	111000	rs1	0	unused (zero)	rs2
10	rd	111000	rs1	1	siconst ₁₃	

• Return from Trap Instructions

Assembler Syntax

rett *address*

Instruction Formats

31:30	29:25	24:19	18:14	13	12:5	4:0
10	00000	111001	rs1	0	unused (zero)	rs2
10	00000	111001	rs1	1	siconst ₁₃	

• Call and Link Instructions

Assembler Syntax

call *label*

Instruction Format

31:30	29:0
01	disp ₃₀

• Restore and Save Instructions

Assembler Syntax

operation *reg_{rs1}*, *reg_{rs2}*, *reg_{rd}*
operation *reg_{rs1}*, *siconst₁₃*, *reg_{rd}*

Instruction Formats

31:30	29:25	24:19	18:14	13	12:5	4:0
10	rd	op3	rs1	0	unused (zero)	rs2
10	rd	op3	rs1	1	siconst ₁₃	

Operations

operation	op3
restore	111101
save	111100

• Trap Instructions

Assembler Syntax

operation *address*

Instruction Formats

31:30	29	28:25	24:19	18:14	13	12:5	4:0
10	F	cond	111010	rs1	0	unused (zero)	rs2
10	F	cond	111010	rs1	1	siconst ₁₃	

Operations

operation	cond	operation	cond
tn	0000	ta	1000
te (tz)	0001	tne (tnz)	1001
tle	0010	tg	1010
tl	0011	tge	1011
tleu	0100	tgu	1100
tcs (tlu)	0101	tcc (tgeu)	1101
tneg	0110	tpos	1110
tvs	0111	tvc	1111

• The NOP Instruction

Assembler Syntax

nop

Instruction Format

31:30	29:25	24:22	21:0
00	00000	100	000000000000000000000000

• The UNIMP Instruction

Assembler Syntax

unimp *const₂₂*

Instruction Format

31:30	29:25	24:22	21:0
00	reserved	000	const ₂₂

• The STBAR Instruction

Assembler Syntax

stbar

Instruction Format

31:30	29:25	24:19	18:14	13:0
10	00000	101000	01111	unused (zero)

• Read State Register Instructions

Assembler Syntax

`rd statereg, rd`

Instruction Format

31:30	29:25	24:19	18:14	13:0
10	rd	op3	rs1	unused (zero)

Register Encodings

state register	op3	rs1
%y	101000	0
%asr1–%asr31	101000	1–31
%psr	101001	reserved
%wim	101010	reserved
%tbr	101011	reserved

• Write State Register Instructions

Assembler Syntax

`wr regrs1, regrs2, statereg`
`wr regrs1, siconst13, statereg`

Instruction Formats

31:30	29:25	24:19	18:14	13	12:5	4:0
10	rd	op3	rs1	0	unused (zero)	rs2
10	rd	op3	rs1	1	siconst ₁₃	

Register Encodings

state register	op3	rd
%y	110000	0
%asr1–%asr31	110000	1–31
%psr	110001	reserved
%wim	110010	reserved
%tbr	110011	reserved

• FLUSH Instructions

Assembler Syntax

`flush address`

Instruction Formats

31:30	29:25	24:19	18:14	13	12:5	4:0
10	unused	111011	rs1	0	unused (zero)	rs2
10	unused	111011	rs1	1	siconst ₁₃	

• Assembler Directives

`.align n`
`.ascii "string" [, "string"]*`
`.asciz "string" [, "string"]*`
`.byte 8-bitval [, 8-bitval]*`
`.data`
`.global label [, label]*`
`.half 16-bitval [, 16-bitval]*`
`.include "filename"`
`.set symbol, 32-bitval`
`.skip n`
`.text`
`.word 32-bitval [, 32-bitval]*`

• ISEM Traps

0 exit (terminate program)
 1 putchar(%r8)
 2 getchar(%r8) (blocking)
 4 putintex(%r8)
 5 getint(%r8)

• Synthetic Instructions

Synthetic instruction	Implementation
<code>bclr rs, rd</code>	<code>andn rd, rs, rd</code>
<code>bclr rs, siconst₁₃</code>	<code>andn rs, siconst₁₃, rd</code>
<code>bset rs, rd</code>	<code>or rd, rs, rd</code>
<code>bset siconst₁₃, rd</code>	<code>or rd, siconst₁₃, rd</code>
<code>btst rs₁, rs₂</code>	<code>andcc rs₁, rs₂, %g0</code>
<code>btst rs, siconst₁₃</code>	<code>andcc rs, siconst₁₃, %g0</code>
<code>btog rs, rd</code>	<code>xor rd, rs, rd</code>
<code>btog rs, siconst₁₃</code>	<code>xor rs, siconst₁₃, rd</code>
<code>call address</code>	<code>jmp_l address, %o7</code>
<code>clr rd</code>	<code>or %g0, %g0, rd</code>
<code>clrb [address]</code>	<code>stb %g0, [address]</code>
<code>clrh [address]</code>	<code>sth %g0, [address]</code>
<code>clr [address]</code>	<code>st %g0, [address]</code>
<code>cmp rs₁, rs₂</code>	<code>subcc rs₁, rs₂, %g0</code>
<code>cmp rs, siconst₁₃</code>	<code>subcc rs, siconst₁₃, %g0</code>
<code>dec rd</code>	<code>sub rd, 1, rd</code>
<code>dec siconst₁₃, rd</code>	<code>sub rd, siconst₁₃, rd</code>
<code>deccc rd</code>	<code>subcc rd, 1, rd</code>
<code>deccc siconst₁₃, rd</code>	<code>subcc rd, siconst₁₃, rd</code>
<code>inc rd</code>	<code>add rd, 1, rd</code>
<code>inc siconst₁₃, rd</code>	<code>add rd, siconst₁₃, rd</code>
<code>inccc rd</code>	<code>addcc rd, 1, rd</code>
<code>inccc siconst₁₃, rd</code>	<code>addcc rd, siconst₁₃, rd</code>
<code>jmp address</code>	<code>jmp_l address, %c0</code>
<code>mov rs, rd</code>	<code>or %g0, rs, rd</code>
<code>mov siconst₁₃, rd</code>	<code>or %g0, siconst₁₃, rd</code>
<code>mov statereg, rd</code>	<code>rd statereg, rd</code>
<code>mov rs, statereg</code>	<code>wr %g0, rs, statereg</code>
<code>mov siconst₁₃, statereg</code>	<code>wr %g0, siconst₁₃, statereg</code>
<code>neg rs, rd</code>	<code>sub %g0, rs, rd</code>
<code>neg rd</code>	<code>sub %g0, rd, rd</code>
<code>not rd</code>	<code>xnor rd, %g0, rd</code>
<code>not rs, rd</code>	<code>xnor rs, %g0, rd</code>
<code>restore</code>	<code>restore %g0, %g0, %g0</code>
<code>save</code>	<code>save %g0, %g0, %g0</code>
<code>ret</code>	<code>jmp_l %i7+8, %g0</code>
<code>retl</code>	<code>jmp_l %o7+8, %g0</code>
<code>set iconst, rd</code>	<code>or %g0, iconst, rd</code>
	<code>—or—</code>
	<code>sethi %hi(iconst), rd</code>
	<code>—or—</code>
	<code>sethi %hi(iconst), rd</code>
	<code>or rd, %lo(iconst), rd</code>
<code>tst rs</code>	<code>orcc %g0, rs, %g0</code>