KASM Assembly Manual

Richard Hughey

UCSC Kestrel Project
Department of Computer Engineering
Jack Baskin School of Engineering
University of California
Santa Cruz, CA

Technical Report UCSC-CRL-98-11
September, 1998

Contents

1 Introduction

This manual documents the opcodes, fields, and directives supported by the new KASM assembler. The manual assumes familiarity with the Kestrel architecture. The reader is referred to the papers available from the http://www.cse.ucsc.edu/research/kestrel for more information on the system architecture.

2 Running Kasm

The KASM command line is:

```
newkasm [-g] [-b] [-Dsym=value] [-o outfile] file[.kasm]
```

Where `file` is a KASM assembly language file. The `-g` option indicates that the output file should be annotated with debugging information. The `-b` option, useful in debugging assembly code, indicates that output should be in binary rather than the hexadecimal required by KESTREL The `[-D]` option can be used to define one or more symbols as if they had `define` statements in the assembly file. The KASM assembler will output Kestrel object code, and debugging information if requested, into `file.ko`. The `-o` option indicates that a different file should be used. Error messages are printed to `stderr` in a format compatible with Emacs' compile mode.

3 Running Kestrel

KASM programs are run in the Kestrel runtime environment. The KESTREL command line is:

```
kestrel -[s|b] [-debug] objectfile.ko inputfile ouputfile [#procesors]
```

Here, the selection of `s` or `b` picks either the simulator or the Kestrel hardware. Use of the Kestrel hardware requires that one of the two Kestrel boards be available and that its socket-based server be running.
The `debug` option places the user immediately in the Kestrel debugger. A series of menus can be used to run or step the program and examine the contents of PEs in either the simulator or the hardware.
The object file is the output from the assembler, and the required input and output files have the data for the program, which must be in the order required by the programs. When KESTREL is used to access the simulator, the number of processing elements can be specified (the default is `#512`, to agree with the hardware).

4 Kasm syntax

KASM assembly code makes use of the opcodes, fields, and directives specified below. A typical KASM line includes one or more compatible opcodes, a destination register, zero to three Kestrel operands, and possibly some instruction modifiers or controller commands. Comments are delimited with semicolons — whenever a semicolon is included in a line, the semicolon and the remainder of the line are ignored.
Opcodes, labels, defines, and macro names are all case insensitive.
Each opcode description includes its operation, syntax, a list (apart from registers) of Kestrel internal state that is used and that is changed, description and comments, and when appropriate a bit pattern. The bit patterns define the behavior of the instruction in the assembler. Bit patters are made up of several single-letter specifiers which are interpreted as follows:

x	Does not affect given bit
0,1	Bit set to 0 or 1
z,w	Bit defaults to 0 or 1 (can be overridden)
A,B	1/0 if opA is first operand, 0/1 otherwise (func bits only)
a,b	Default A,B (can be overridden)
E	Either opA or opB must be specified but not both (opA and opB bits only)
e	Default E (can be overridden)
l	Label required in Cimm
r	Required — must be specified by another field
u	Unused bit, error if another field sets

Opcodes can generally be listed in any order on a line, except that those that require operands should be listed first to ensure the correct parsing of the operands. For example, to perform an add-min instruction, the code

```
add L1, L2, MDR, L3, minc
```

would generate an error message because the third source operand (`L3`) is unneeded for an addition. The correct form would place the `minc` after the `add`.
It is suggested opcodes and fields that require operands or affect the output of the ALU (such as `mp`, the arithmetic multi-precision indicator) be placed first on the line, followed by the operands, followed by other modifiers, such controller (for example, `jump`) and bitshifter control fields (for example, `bspush`).
Comments, symbolic constants (see `define`), and macros can be used to make code more readable.

5 Instruction operands

bs — Bit shifter opB — bs

Operation: opB ← bs **Uses:** BS
Syntax: `bs` **Sets:** OpB

Description: Select the bitshifter as operand B.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	110	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

destreg — Register destination — destreg

Operation: dest is Register **Uses:** –
Syntax: `destreg` dest **Sets:** –

Description: Select a register (specified as L0-L31 or R0-R31) for the destination. All instructions write to a destination register, which defaults to register L0 if not specified. The destination is always the first register on a line of code.
Comments: The DESTREG specifier is optional

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	rrrrrr	xxxxxxxx

mdr — MDR opB — mdr

Operation: opB ← MDR **Uses:** Mdr
Syntax: `mdr` **Sets:** OpB

Description: Select the SRAM's memory data register as operand B.
Comments: Use READ on a previous instruction to load the MDR from SRAM.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	010	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

mhi — Multhi opB — mhi

Operation: opB ← mhi **Uses:** MHi
Syntax: `mhi` **Sets:** OpB

Description: Set opB to multiplier high byte register mhi, the result of a previous multiply.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	100	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

opareg — Register opA — opareg

Operation: opA ← Register **Uses:** –
Syntax: `opareg` opA **Sets:** –

Description: Select a register (specified as L0-L31 or R0-R31) for operand A.
Comments: The OPAREG specifier is optional.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	xxx	rrrrrr	xxxxxx	xxxxxx	xxxxxxxx

opbimm — Immediate opB — opbimm

Operation: opB ← #nnnn **Uses:** –
Syntax: `opbimm #Imm` **Sets:** OpB

Description: Select an immediate as operand B. The immediate can be a signed (-128 to 127) or unsigned decimal (0 to 255), unsigned octal (0000 to 0377), or unsigned hexadecimal (0x00 to 0xff). The immediate is proceeded by a pound sign (#) to distinguish it from the controller immediate. See also SCRTOIMM.
Comments: The OPBIMM specifier is optional.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	111	xxxxxx	xxxxxx	xxxxxx	rrrrrrrr

opbreg — Register opB — opbreg

Operation: opB ← opC **Uses:** –
Syntax: `opbreg` opC **Sets:** OpB

Description: Select a register (specified as L0-L31 or R0-R31) for operand B. If operand C is used in this instruction, the same register must be specified for operand C and operand B. Because of this restriction, the assembler prefers to assign register operands to operand A rather than operand B where possible.
Comments: The OPBREG specifier is optional.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	000	xxxxxx	rrrrrr	xxxxxx	xxxxxxxx

opbsreg — Sign extend Register opB — opbsreg

Operation: opB ← opC[7] **Uses:** –
Syntax: `opbsreg` opC **Sets:** OpB

Description: Select the sign extension of a register (specified as SL0-SL31 or SR0-SR31) for operand B. If operand C is used in this instruction, the same register must be specified for operand C and operand B.
Comments: The OPBSREG specifier is optional.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	001	xxxxxx	rrrrrr	xxxxxx	xxxxxxxx

opcreg — Register opC — opcreg

Operation: opC ← Register **Uses:** –
Syntax: `opcreg` opA **Sets:** –

Description: Select a register (specified as L0-L31 or R0-R31) for operand C. This register may also be accessed or used as operand B.
Comments: The OPCREG specifier is optional

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	xxx	rrrrrr	xxxxxx	xxxxxx	xxxxxxxx

smdr — Sign extend MDR opB — smdr

Operation: opB ← SgnEx (MDR[7]) **Uses:** Mdr
Syntax: `smdr` **Sets:** OpB

Description: Select the sign extension of the memory data register as operand B.
Comments: Use READ on a previous instruction to load the MDR from SRAM.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	011	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

smhi — Sign extend multhi opB — smhi

Operation: opB ← SgnEx (mhi[7]) **Uses:** MHi
Syntax: `smhi` **Sets:** OpB

Description: Set opB to sign extension of the multiplier high byte, the result of a previous multiply.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	101	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

6 Logical Instructions

AND — And — AND

Operation: RESULT ← OPB ∧ OPA **Uses:** –
Syntax: `AND` DEST, OPB, OPA **Sets:** –

Description: Logical and.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	00111	0	0	0	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

INVERT — Invert Operand — INVERT

Operation: RESULT ← ¬OP1 **Uses:** –
Syntax: `INVERT` DEST, [OPA *or* OPB] **Sets:** –

Description: The 1s complement of operand A or B.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	01BA0	0	0	0	x	xxxxx	xxxx	xx	x	x	EEE	EEEEEE	xxxxxx	rrrrrr	xxxxxxxx

MOVE — Move — MOVE

Operation: RESULT ← OP1 **Uses:** –
Syntax: `MOVE` DEST, [OPA *or* OPB] **Sets:** –

Description: Move operand A or operand B to destination.
Comments: Register OpC and can be used with SRAM or for comparison as move takes place.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	00AB1	0	0	0	x	xxxxx	xxxx	xx	x	x	EEE	EEEEEE	xxxxxx	rrrrrr	xxxxxxxx

MOVEC — Move Operand C — MOVEC

Operation: RESULT ← OPC **Uses:** –
Syntax: `MOVEC` DEST, OPC **Sets:** –

Description: Move register operand C to destination.
Comments: This should only be used if the ALU is producing a flag or state of interest at the same time. For example, MOVEC SUB L2,L3,MDR,R2,fbats,push will, at the same time as copying R2 to L2 via operand C, subtract the MDR from L3, place that sign on the flag bus and push that bit onto the bitshifter, turning off PEs for which L3 is less than the MDR. SRAM base plus register addressing is not available during a MOVEC.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	10	x	x	xxx	xxxxxx	rrrrrr	rrrrrr	xxxxxxxx

NAND — Nand — NAND

Operation: RESULT ← ¬ (OPA ∧ OPB) **Uses:** –
Syntax: `NAND` DEST, OPB, OPA **Sets:** –

Description: Bitwise NAND function.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	01000	0	0	0	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

NOP

NOP

Operation: – **Uses:** –
Syntax: `NOP` **Sets:** –

Description: No Kestrel array operation.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	00101	0	0	0	z	xwxxx	zzzz	zz	z	z	zzz	000000	000000	000000	xxxxxxxx

NOR

Nor

Operation: RESULT $\leftarrow \neg($opA $\vee$ opB $)$ **Uses:** –
Syntax: `NOR` DEST, opB, opA **Sets:** –

Description: Bitwise NOR function.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	01110	0	0	0	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

OR

Or

Operation: RESULT $\leftarrow$ opA $\vee$ opB **Uses:** –
Syntax: `OR` DEST, opB, opA **Sets:** –

Description: Bitwise OR function.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	00001	0	0	0	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

XNOR

Exclusive NOR

Operation: RESULT $\leftarrow \neg$ (opA $\oplus$ opB) **Uses:** –
Syntax: `XNOR` DEST, opB, opA **Sets:** –

Description: Bitwise exclusive NOR.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	00110	0	0	0	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

XOR

Exclusive OR

Operation: RESULT $\leftarrow$ opA $\oplus$ opB **Uses:** –
Syntax: `XOR` DEST, opB, opA **Sets:** –

Description: Bitwise exclusive OR.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	01001	0	0	0	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

7 Arithmetic Instructions

ADD — Add — ADD

Operation: RESULT ← opB + opA **Uses:** –
Syntax: `ADD` DEST, opB, opA **Sets:** Carry

Description: Addition of two bytes. Use ADD MP on high bytes for multiprecision addition. Use ADD C1 for low byte of A + B + 1.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	10001	z	z	1	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

ADDXX — Add, Operand to self — ADDXX

Operation: RESULT ← op1 + op1 **Uses:** –
Syntax: `ADDXX` DEST, [opA *or* opB] **Sets:** Carry

Description: Add operand A or B to itself. Use ADDXX C1 to add 1 more and ADDXX MP for higher bytes.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	10AB1	z	z	1	x	xxxxx	xxxx	xx	x	x	EEE	EEEEEE	xxxxxx	rrrrrr	xxxxxxxx

ADDXZ — Add, Zero and Operand — ADDXZ

Operation: RESULT ← op1 + 0 **Uses:** –
Syntax: `ADDXZ` DEST, [opA *or* opB] **Sets:** Carry

Description: Add operand A or B to zero. Use ADDXZ C1 to increment with ADDXZ MP for the higher bytes. Use ADDXZ MP to add the carry latch to the operand.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	00AB1	z	z	1	x	xxxxx	xxxx	xx	x	x	EEE	EEEEEE	xxxxxx	rrrrrr	xxxxxxxx

ADDZZ — Add, Zero — ADDZZ

Operation: RESULT ← 0 + 0 **Uses:** –
Syntax: `ADDZZ` DEST **Sets:** Carry

Description: Copy 0 to destination. Use ADDZZ C1 for 1 and ADDZZ MP to copy the carry latch to the destination.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	01111	z	z	1	x	xxxxx	xxxx	xx	x	x	uuu	uuuuuu	xxxxxx	rrrrrr	xxxxxxxx

mp — Arithmetic Multiprecision Indicator — mp

Operation: MP ← 1 **Uses:** Carry
Syntax: `mp` **Sets:** Carry

Description: Set the ALU to use multiprecision mode. Does not affect multiplier or comparator operation.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	rrrrr	0	1	1	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

SUB — Subtract, op1 and op2 — SUB

Operation: RESULT ← OP1 − OP2 **Uses:** –
Syntax: `SUB` DEST, OPB, OPA **Sets:** Carry

Description: Subtract operands A and B in either order.
Comments: Use SUB for low byte of multiprecision subtract, SUB MP for higher bytes. Use SUB B1 to subtract an additional 1.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	10AB0	w	z	1	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

SUBXZ — Subtract, Operand and Zero — SUBXZ

Operation: RESULT ← OP1 − 0 **Uses:** –
Syntax: `SUBXZ` DEST, [OPA *or* OPB] **Sets:** Carry

Description: Subtract Zero from operand A or B. Use SUBXZ MP for higher bytes and SUBXZ B1 to decrement.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	11AB1	w	z	1	x	xxxxx	xxxx	xx	x	x	EEE	EEEEEE	xxxxxx	rrrrrr	xxxxxxxx

SUBZX — Subtract, Zero and Operand — SUBZX

Operation: RESULT ← 0 − OP1 **Uses:** –
Syntax: `SUBZX` DEST, [OPA *or* OPB] **Sets:** Carry

Description: Subtract operand A or B from Zero. Use SUBZX MP for higher bytes. Use SUBZX B1 to subtract one more.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	01AB0	w	z	1	x	xxxxx	xxxx	xx	x	x	EEE	EEEEEE	xxxxxx	rrrrrr	xxxxxxxx

SUBZZ — Subtract, Zero — SUBZZ

Operation: RESULT ← 0 − 0 **Uses:** –
Syntax: `SUBZZ` DEST **Sets:** Carry

Description: Copy 0 to destination. Use SUBZZ B1 for -1 and SUBZZ MP to place -1 in the destination if there was a borrow on the previous subtract.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	00000	w	z	1	x	xxxxx	xxxx	xx	x	x	uuu	uuuuuu	xxxxxx	rrrrrr	xxxxxxxx

8 Multiplier Instructions

addmc — Multiply, Adding C — addmc

Operation: (Mhi,dest) ← Product + opC **Uses:** –
Syntax: `addmc` opC **Sets:** MHi

Description: Add unsigned register operand C to the product of a multiply. Modifier for MULT, MULSA, MULTSB, and MULTSAB.
Comments: This is used to bring down partial products from the previous row of multiplication in a multi-precision multiply. For example, in multiplying (1,2,3) by (4,5,6), the results of (1,2,3) by (6) are stored in registers. In multiplying (1,2,3) by (5), the appropriate registers are added back into the multiplication.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	1rrrz	1	1	0	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	rrrrrr	xxxxxx	xxxxxxxx

addmhi — Multiply, Adding MHi — addmhi

Operation: (Mhi,dest) ← Product + Mhi **Uses:** MHi
Syntax: `addmhi` **Sets:** MHi

Description: Add unsigned register Mhi to the product of a multiply. Modifier for MULT, MULSA, MULTSB, and MULTSAB.
Comments: This is used to chain partial products in the current row in a multi-precision multiply. For example, in multiplying (1,2,3) by (4,5,6), the (3) by (6) multiply is a basic MULT, while the (2) by (6) multiply used ADDMHI to add in the high byte of the just-performed multiply. The (3) by (5) multiply will use ADDMC to add in the second byte of (1,2,3) by (6), while the (2) by (5) multiply will use both ADDMC to add in the third byte of the previous row and ADDMHI to add in the high byte of the (3) by (5) multiply.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	zrrr1	1	1	0	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

MULT — Multiply — MULT

Operation: (Mhi,dest) ← opA × opB **Uses:** –
Syntax: `MULT` dest`,` opB`,` opA **Sets:** MHi

Description: Multiply two unsigned bytes to produce a two-byte result.
Comments: Use for the first multiply of signed or unsigned multibyte multiplies. Use with ADDMHI and ADDMC for all partial products not involving a signed byte.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	z100z	1	1	0	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

MULTSA — Multiply, signed A — MULTSA

Operation: (Mhi,dest) ← ± opA × opB **Uses:** –
Syntax: `MULTSA` dest`,` opB`,` opA **Sets:** MHi

Description: Multiply a signed operand A with an unsigned operand B to produce a 2-byte result.
Comments: Use in place of MULT when dealing the sign byte (MSB) of a signed operand A and an unsigned byte of operand B.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	z101z	1	1	0	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

MULTSAB **Multiply, signed A and B** MULTSAB

Operation: (MHI,DEST) ← ± opA × ± opB **Uses:** –

Syntax: `MULTSAB` DEST, opB, opA **Sets:** MHI

Description: Multiply two signed operands A and B to produce a 2-byte result.
Comments: Use in place of MULT when multiplying the sign bytes (MSBs) of signed operands A and B.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	z111z	1	1	0	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

MULTSB **Multiply, signed B** MULTSB

Operation: (MHI,DEST) ← opA × ± opB **Uses:** –

Syntax: `MULTSB` DEST, opB, opA **Sets:** MHI

Description: Multiply an unsigned operand A with a signed operand B to produce a 2-byte result.
Comments: Use in place of MULT when dealing with the sign byte (MSB) of a signed operand B and an unsigned byte of operand A.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	z110z	1	1	0	x	xxxxx	xxxx	xx	x	x	rrr	rrrrrr	xxxxxx	rrrrrr	xxxxxxxx

9 Selection Instructions

MAXC — Maximize with C — MAXC

Operation: DEST ← MAX(RESULT, OPC) **Uses:** –
Syntax: `MAXC` DEST`, [`OPA *or* OPB`], `OPC **Sets:** CMP

Description: Select the maximum of the result and operand C. Use on top byte of multiprecision operands (MAXC CMP on lower bytes). May be used with two operands by itself or with an ALU instruction.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	zzabw	x	x	x	0	00000	xxxx	11	x	x	eee	eeeeee	rrrrrr	rrrrrr	xxxxxxxx

MINC — Minimize with C — MINC

Operation: DEST ← MIN(RESULT, OPC) **Uses:** –
Syntax: `MINC` DEST`, [`OPA *or* OPB`], `OPC **Sets:** CMP

Description: Select the minimum of the result and operand C. Can use directly with a single operand or combine with an ALU instruction as in ADD MINC dest, op1, op2, opC. For multiprecision addmin, use ADD on the lowest bytes saving to a register, ADD MP on the middle bytes saving to a register, ADD MP MINC on the top byte, and MINC CMP on the saved registers and lower bytes of the comparison operand.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	zzabw	z	z	z	1	00000	xxxx	11	x	x	eee	eeeeee	rrrrrr	rrrrrr	xxxxxxxx

MODMAXC — Maximize with C, Mod 246 — MODMAXC

Operation: DEST ← MODMAX(RESULT, OPC) **Uses:** –
Syntax: `MODMAXC` DEST`, [`OPA *or* OPB`], `OPC **Sets:** CMP

Description: Select the mod-256 maximum of the result and operand C. Use on top byte of multiprecision operands with MODMAXC CMP on lower bytes. May used with two operands by itself or with an ALU instruction.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	zzabw	x	x	x	0	00010	xxxx	11	x	x	eee	eeeeee	rrrrrr	rrrrrr	xxxxxxxx

MODMINC — Minimize with C, Mod 256 — MODMINC

Operation: DEST ← MODMIN(RESULT, OPC) **Uses:** –
Syntax: `MODMINC` DEST`, [`OPA *or* OPB`], `OPC **Sets:** CMP

Description: Select the mod-256 minimum of the result and operand C. Use on top byte of multiprecision operands with MODMINC CMP on lower bytes. May be used with two operands by itself or with an ALU instruction.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	zzabw	x	x	x	1	00010	xxxx	11	x	x	eee	eeeeee	rrrrrr	rrrrrr	xxxxxxxx

SELECTC — Select — SELECTC

Operation: DEST ← (FLAG=0) ? OPC : RESULT **Uses:** –
Syntax: `SELECTC` DEST`,` `[`OPA *or* OPB`]``,` OPC **Sets:** –

Description: Asserted low selection. Select between operand C and the result according to the specified flag. May be used with two operands by itself or with an ALU instruction.
Comments: Select may be done top-down or bottom-up. Selection between (1,2)+(3,4) and (5,6) into (7,8) requires three instructions: ADD 8,2,4 followed by ADD MP SELECT 7,1,3,5 followed by SELECT 8,8,6. The chosen flag must be specified with each selection command, and (3,4) is a non-register operand B. FBINV may be used to change the polarity of the selection.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	zzabw	x	x	x	z	rrrrr	xxxx	11	x	x	eee	eeeeee	rrrrrr	rrrrrr	xxxxxxxx

SMAXC — Maximize with C, Signed — SMAXC

Operation: DEST ← MAX(RESULT, OPC) **Uses:** –
Syntax: `SMAXC` DEST`,` `[`OPA *or* OPB`]``,` OPC **Sets:** CMP

Description: Select the maximum of the signed result and signed operand C. Use on top byte of multi-precision operands with SMAXC CMPSWAP with swapped register operands on lower bytes. May be used with two operands by itself or with an ALU instruction.
Comments: See comment for CMPSWAP on operand swaping.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	zzabw	x	x	x	1	00001	xxxx	11	x	x	eee	eeeeee	rrrrrr	rrrrrr	xxxxxxxx

SMINC — Minimize with C, Signed — SMINC

Operation: DEST ← MIN(RESULT, OPC) **Uses:** –
Syntax: `SMINC` DEST`,` `[`OPA *or* OPB`]``,` OPC **Sets:** CMP

Description: Select the minimum of the signed result and signed operand C. Use on top byte of multiprecision operands and SMINC CMPSWAP with swapped register operands on the lower bytes. May be used with two operands by itself or with an ALU instruction.
Comments: See comment for CMPSWAP on operand swapping.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	zzabw	x	x	x	0	00001	xxxx	11	x	x	eee	eeeeee	rrrrrr	rrrrrr	xxxxxxxx

10 Comparison Instructions

cmp — Comparator multiprecision — cmp

Operation: **Uses:** –
Syntax: `cmp` **Sets:** Cmp

Description: (Calls internal kasm function.) Use for low bytes of a multiprecision selection or comparison instruction.
Comments: Cannot be used with signed comparison operations; instead use CMPSWAP.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	w	00xxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

cmpswap — Comparator multiprecision — cmpswap

Operation: **Uses:** –
Syntax: `cmpswap` **Sets:** Cmp

Description: (Calls internal kasm function.) Use for low bytes of a multiprecision selection or comparison instruction.
Comments: Can only be used with signed min/max/comparison operation. The low byte operands must be in the opposite order from the high byte operands. In general, only signed register-register comparisons can be performed because the various operand B alternatives (mdr, immediate, etc) cannot be used as an operand A. For example, if taking the signed maximum of (2,1) and (4,3) into (8,7), the appropriate instructions are SMAXC 8,2,4 and SMAXC CMPSWAP 7,3,1.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	w	00xxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

EQUALC — Equal to C — EQUALC

Operation: FLAG ← 0 IF RESULT $< C$ **Uses:** Cmp
Syntax: `EQUALC [`opA *or* opB`]`, opC **Sets:** –

Description: Asserted-low unsigned comparison check. Single precision. For multiprecision, perform a multiprecision LTC/LTC MP on the data and on the next instruction use EQLATCH to put the equality latch on the flag bus and FBINV to make it asserted-low. Use with MOVE or other ALU instruction.
Comments: This instruction is geared for use on the bitshifter. A push will maintain activity in all PEs for which the condition holds. The flag bus can be inverted with FBINV to directly set the MASK, save an asserted-high value, or complement the test.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	rrrrr	x	x	x	w	10000	xxxx	xx	x	x	eee	eeeeee	rrrrrr	xxxxxx	xxxxxxxx

LTC — Less than C — LTC

Operation: FLAG ← 0 IF RESULT $< C$ **Uses:** –
Syntax: `LTC [`opA *or* opB`]`, opC **Sets:** Cmp

Description: Asserted-low unsigned comparison check. Use for first (top) byte, with LTC CMP for lower bytes. Can be used with MOVE to compare two operands or with an arithmetic instruction such as ADD.
Comments: This instruction is geared for use on the bitshifter. A push will maintain activity in all PEs for which the condition holds. The flag bus must be inverted with FBINV to directly set the MASK or save an asserted-high value. Use FBINV for greater than or equal. To perform a less-than-or-equal check onto the bitshifter, do, for example, LTC PUSH followed by NOP EQLATCH FBINV BSOR

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	rrrrr	x	x	x	z	00000	xxxx	xx	x	x	eee	eeeeee	rrrrrr	xxxxxx	xxxxxxxx

SLTC — Signed Less than C — SLTC

Operation: FLAG ← 0 IF RESULT $< C$ **Uses:** –

Syntax: `SLTC [`OPA *or* OPB`],` OPC **Sets:** CMP

Description: Asserted-low signed comparison check. Use for first (top) byte, with SLTC CMP for lower bytes (SLTC CMPSWAP, with operands swapped, if FBINV is set). Use with MOVE or other ALU instruction.

Comments: This instruction is geared for use on the bitshifter. A push will maintain activity in all PEs for which the condition holds. The flag bus must be inverted with FBINV to directly set the MASK or save an asserted-high value. Use FBINV for greater than or equal. To perform a less-than-or-equal check onto the bitshifter, do, for example, SLTC PUSH followed by NOP EQLATCH FBINV BSAND

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	rrrrr	x	x	x	w	00001	xxxx	xx	x	x	eee	eeeeee	rrrrrr	xxxxxx	xxxxxxxx

11 Bitshifter Modifiers

bsand — And bitshifter — bsand

Operation: BS[7] ← BS[7] ∧ FLAG, MASK ← NOR(BS) **Uses:** BS
Syntax: `bsand` **Sets:** BS,MSK

Description: And the flag bit into the MSB of the bitshifter. Both the flag and BS[7] are asserted LOW for this operation.
Comments: Occurs in all PEs. Mask bit is changed.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	rrrrr	0101	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bsclear — Clear bitshifter — bsclear

Operation: BS ← 0, MASK ← MASK **Uses:** –
Syntax: `bsclear` **Sets:** BS

Description: Clear the bitshifter.
Comments: Occurs in all PEs. Mask bit is NOT changed.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	1000	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bsclearm — Clear bitshifter set mask — bsclearm

Operation: BS ← 0, MASK ← 1 **Uses:** BS
Syntax: `bsclearm` **Sets:** BS,MSK

Description: Clear the bitshifter.
Comments: Occurs in all PEs. Mask bit is changed.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	1001	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bscondlatch — Conditional bitshifter latch — bscondlatch

Operation: BS ← RESULT **Uses:** BS
Syntax: `bscondlatch` **Sets:** BS

Description: Conditionally latch the instruction result in the bitshifter.
Comments: Occurs in PEs with asserted Mask flag.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	1101	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bscondleft — Conditional left shift — bscondleft

Operation: BS[7:0] ← (BS[6:0],FLAG) **Uses:** BS
Syntax: `bscondleft` **Sets:** BS

Description: Left-shift the bitshifter, appending the flag.
Comments: Occurs only in unmasked PEs.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	rrrrr	0011	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bscondright — Conditional right shift — bscondright

Operation: BS[7:0] ← (FLAG,BS[7:1]) **Uses:** BS
Syntax: `bscondright` **Sets:** BS

Description: Right-shift the bitshifter, prepending the flag.
Comments: Occurs only in unmasked PEs.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	rrrrr	1111	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bsflagmask — Flag Mask — bsflagmask

Operation: Mask ← FLAG **Uses:** –
Syntax: `bsflagmask` **Sets:** Msk

Description: Copy the specified flag bit to the mask register.
Comments: Occurs in all PEs regardless of Mask value.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	rrrrr	0010	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bslatch — Unconditional bitshifter latch — bslatch

Operation: BS ← RESULT, Mask ← NOR(BS) **Uses:** BS
Syntax: `bslatch` **Sets:** BS,Msk

Description: Unconditionally latch the instruction result in the bitshifter.
Comments: Occurs in all PEs. Mask bit is changed.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	1100	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bsnot — Not bitshifter — bsnot

Operation: BS[7] ← ¬BS[7], Mask ← NOR(BS) **Uses:** BS
Syntax: `bsnot` **Sets:** BS,Msk

Description: Complement the MSB of the bitshifter. Use for Else clause.
Comments: Occurs in all PEs. Mask bit is changed.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	0110	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bsnotmask — Not Mask — bsnotmask

Operation: Mask ← ¬Mask **Uses:** Msk
Syntax: `bsnotmask` **Sets:** Msk

Description: Complement the mask flag.
Comments: Occurs in all PEs regardless of Mask value.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	0001	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bsor — Or bitshifter — bsor

Operation: BS[7] ← BS[7] ∨ FLAG, MASK ← NOR(BS) **Uses:** BS
Syntax: `bsor` **Sets:** BS,MSK

Description: Or the flag bit into the MSB of the bitshifter. Both the flag and BS[7] are asserted LOW for this operation.
Comments: Occurs in all PEs. Mask bit is changed.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	rrrrr	0100	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bspop — Pop bitshifter — bspop

Operation: BS[7:0] ← (BS[6:0],0), MASK ← NOR(BS) **Uses:** BS
Syntax: `bspop` **Sets:** BS,MSK

Description: Shift bitshifter. Use for complete a conditional nesting level.
Comments: Occurs in all PEs. Mask bit is changed.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	1011	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bspopnot — Not bitshifter — bspopnot

Operation: BS[7:0] ← (! BS[6],BS[5:0],0), MASK ← NOR(BS) **Uses:** BS
Syntax: `bspopnot` **Sets:** BS,MSK

Description: Shift bitshifter and Complement the new MSB of the bitshifter. Use for Else clause at one-greater nesting level.
Comments: Occurs in all PEs. Mask bit is changed.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	1010	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bspush — Push bitshifter — bspush

Operation: BS[7:0] ← (FLAG, BS[7:1]), MASK ← NOR(BS) **Uses:** BS
Syntax: `bspush` **Sets:** BS,MSK

Description: Push a new condition on the bitshifter and reset the Mask.
Comments: Occurs in all PEs. Mask bit is changed.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	rrrrr	1110	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

bsset — Set bitshifter — bsset

Operation: BS[7] ← FLAG, MASK ← NOR(BS) **Uses:** BS
Syntax: `bsset` **Sets:** BS,MSK

Description: Set the MSB of the bitshifter. Use for Else clause.
Comments: Occurs in all PEs. Mask bit is changed.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	rrrrr	0111	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

12 Flag Modifiers

fbaco — ALU carry out — fbaco

Operation: FLAG ← ACO **Uses:** –
Syntax: `fbaco` **Sets:** –

Description: Set flag bus to the carry out from the ALU.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	xxxxx	x	x	x	x	01101	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbats — ALU true sign — fbats

Operation: FLAG ← ATS **Uses:** –
Syntax: `fbats` **Sets:** –

Description: Set flag bus to ALU's true sign output.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	xxxxx	x	x	x	x	01100	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbbs0 — Bitshifter 0 — fbbs0

Operation: FLAG ← BS0 **Uses:** BS
Syntax: `fbbs0` **Sets:** –

Description: Set flag bus to the LSB of the bitshifter.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	xxxxx	x	x	x	x	01001	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbbs7 — Bitshifter 7 — fbbs7

Operation: FLAG ← BS7 **Uses:** BS
Syntax: `fbbs7` **Sets:** –

Description: Set flag bus to the MSB of the bitshifter.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	xxxxx	x	x	x	x	01010	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbbsnor — Bitshifter NOR — fbbsnor

Operation: FLAG ← BSNOR **Uses:** BS
Syntax: `fbbsnor` **Sets:** –

Description: Set flag bus to the NOR of the bitshifter.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	xxxxx	x	x	x	x	01000	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbcbo — Comparator borrow out — fbcbo

Operation: FLAG ← CBO **Uses:** CMP
Syntax: `fbcbo` **Sets:** CMP

Description: Set flag bus to comparator's borrow out.

fr ce	alu func	c i	m p	l c	f i	flag bus	bit shift	res mx	sram r	sram w	opB sel	opA register	opC register	dest register	array immediate
x	xxxxx	x	x	x	x	00100	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbclatch **Carry latch** fbclatch

Operation: FLAG ← CLATCH
Syntax: `fbclatch`
Uses: Carry
Sets: –

Description: Set flag bus to ALU's carry latch.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	01110	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbcmsb **Comparator MSB** fbcmsb

Operation: FLAG ← CMSB
Syntax: `fbcmsb`
Uses: Cmp
Sets: Cmp

Description: Set flag bus to the comparator's MSB.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	00110	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbcts **Comparator true sign** fbcts

Operation: FLAG ← CTS
Syntax: `fbcts`
Uses: Cmp
Sets: Cmp

Description: Set flag bus to comparator true sign.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	00101	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbeq **Equal** fbeq

Operation: FLAG ← EQ
Syntax: `fbeq`
Uses: Cmp
Sets: –

Description: Set flag bus to the comparators equal output.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	10000	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbeqlatch **Eqlatch** fbeqlatch

Operation: FLAG ← EQLatch
Syntax: `fbeqlatch`
Uses: Cmp
Sets: –

Description: Set flag bus to the comparator's equality latch .

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	10001	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshd16 **Meshd16** fbmeshd16

Operation: FLAG ← MESHD16
Syntax: `fbmeshd16`
Uses: BS
Sets: –

Description: Set flag bus to meshd16.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	11011	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshd32 **Meshd32** fbmeshd32

Operation: FLAG ← MESHD32 **Uses:** BS
Syntax: `fbmeshd32` **Sets:** –

Description: Set flag bus to meshd32.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	11111	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshd8 **Meshd8** fbmeshd8

Operation: FLAG ← MESHD8 **Uses:** BS
Syntax: `fbmeshd8` **Sets:** –

Description: Set flag bus to meshd8.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	10111	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshl1 **Meshl1** fbmeshl1

Operation: FLAG ← MESHL1 **Uses:** BS
Syntax: `fbmeshl1` **Sets:** –

Description: Set flag bus to meshl1.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	10100	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshl2 **Meshl2** fbmeshl2

Operation: FLAG ← MESHL2 **Uses:** BS
Syntax: `fbmeshl2` **Sets:** –

Description: Set flag bus to meshl2.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	11000	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshl4 **Meshl4** fbmeshl4

Operation: FLAG ← MESHL4 **Uses:** BS
Syntax: `fbmeshl4` **Sets:** –

Description: Set flag bus to meshl4.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	11100	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshr1 **Meshr1** fbmeshr1

Operation: FLAG ← MESHR1 **Uses:** BS
Syntax: `fbmeshr1` **Sets:** –

Description: Set flag bus to meshr1.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	10101	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshr2 — Meshr2 — fbmeshr2

Operation: FLAG ← MESHR2 **Uses:** BS
Syntax: `fbmeshr2` **Sets:** –

Description: Set flag bus to meshr2.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	11001	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshr4 — Meshr4 — fbmeshr4

Operation: FLAG ← MESHR4 **Uses:** BS
Syntax: `fbmeshr4` **Sets:** –

Description: Set flag bus to meshr4.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	11101	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshu16 — Meshu16 — fbmeshu16

Operation: FLAG ← MESHU16 **Uses:** BS
Syntax: `fbmeshu16` **Sets:** –

Description: Set flag bus to meshu16.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	11010	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshu32 — Meshu32 — fbmeshu32

Operation: FLAG ← MESHU32 **Uses:** BS
Syntax: `fbmeshu32` **Sets:** –

Description: Set flag bus to meshu32.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	11110	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbmeshu8 — Meshu8 — fbmeshu8

Operation: FLAG ← MESHU8 **Uses:** BS
Syntax: `fbmeshu8` **Sets:** –

Description: Set flag bus to meshu8.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	10110	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbminlatch — Min latch — fbminlatch

Operation: FLAG ← MINLATCH **Uses:** CMP
Syntax: `fbminlatch` **Sets:** –

Description: Set flag bus to comparator's min latch.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	00011	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbwor — Wired-or — fbwor

Operation: FLAG ← WOR **Uses:** BS

Syntax: `fbwor` **Sets:** –

Description: Set flag bus to wired-or of all PEs in the same group of 64.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	01011	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

13 Other Array Modifiers

b0 — Borrow-in 0 — b0

Operation: $\textsc{Cin} \leftarrow \neg\ 0$ **Uses:** –
Syntax: `b0` **Sets:** –

Description: Use a borrow-in of 0 (equivalent to c1)

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	1	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

b1 — Borrow-in 1 — b1

Operation: $\textsc{Cin} \leftarrow \neg\ 1$ **Uses:** –
Syntax: `b1` **Sets:** –

Description: Use a borrow-in of 1 (equivalent to c0)

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	0	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

c0 — Carry-in 0 — c0

Operation: $\textsc{Cin} \leftarrow 0$ **Uses:** –
Syntax: `c0` **Sets:** –

Description: Set the ALU's carry-in to 0

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	0	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

c1 — Carry-in 1 — c1

Operation: $\textsc{Cin} \leftarrow 1$ **Uses:** –
Syntax: `c1` **Sets:** –

Description: Set the ALU's carry-in to 1

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	1	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

fbinv — Flag invert — fbinv

Operation: $\textsc{flag} \leftarrow \neg\textsc{flag}$ **Uses:** –
Syntax: `fbinv` **Sets:** –

Description: (Calls internal kasm function.) Invert the flag bus value from that defined by the associated instruction.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	w	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

force — Force operation — force

Operation: **Uses:** –
Syntax: `force` **Sets:** –

Description: Require all PEs to perform the given instructions regardless of Mask flag or BeginCond/EndCond state.
Comments: Bitshifter operations geared to conditional processing include an implicit force for the bitshifter operation.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
1	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	xxxxxxxx

read — Read SRAM — read

Operation: MDR ← SRAM[ADDRESS] **Uses:** –
Syntax: `read #IMM` **Sets:** MDR

Description: (Calls internal kasm function.) Perform a read from the local memory. Address can be of several forms: `read (#nnn)`, `read (L3)`, or `read (#nnn + L3)`, where L3 can be any register. Memory addressing always uses the immediate field, so the second form includes an implicit `+ #0`. When a register is specified in an address, it is operand C. Thus, memory operations are generally not performed with comparator instructions. The data retrieved is placed in the memory data register for use during subsequent instructions.
Comments: Only one SRAM operation (read or write) is permitted in an instruction.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	1	z	xxx	xxxxxx	xxxxxx	xxxxxx	rrrrrrrr

write — Write SRAM — write

Operation: SRAM[ADDRESS] ← RESULT **Uses:** –
Syntax: `write #IMM` **Sets:** SRAM

Description: (Calls internal kasm function.) Perform a write to the local memory. Address can be of several forms: `write (#nnn)`, `write (L3)`, or `write (#nnn + L3)`, where L3 can be any register. Memory addressing always uses the immediate field, so the second form includes an implicit `+ #0`. When a register is specified in an address, it is operand C. Thus, memory operations are generally not performed with comparator instructions.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	z	1	xxx	xxxxxx	xxxxxx	xxxxxx	rrrrrrrr

14 Controller Instructions

arrtoq — Array to Q Out — arrtoq

Operation: QOUT, SCRATCH ← ARRAY **Uses:** –
Syntax: `arrtoq` DEST **Sets:** –

Description: Write the array's output to the output queue and the controller's scratch register. If the destination register is a left register, the output value is the value written by the leftmost processing element and register bank. If the destination is a right register, the output value is from the rightmost processing element.
Comments: If the outputting PE is masked, no output is produced.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	rxxxxx	xxxxxxxx

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	1	x	x	x	1	00	x	x	xx	x	x	x	x	x	x	xx

arrtoscr — Array to Scratch — arrtoscr

Operation: SCRATCH ← ARRAY **Uses:** –
Syntax: `arrtoscr` DEST **Sets:** –

Description: Write the array's output to the controller's scratch register. If the destination register is a left register, the output value is the value written by the leftmost processing element and register bank. If the destination is a right register, the output value is from the rightmost processing element.
Comments: If the outputting PE is masked, no output is produced.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	rxxxxx	xxxxxxxx

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	0	x	x	x	1	00	x	x	xx	x	x	x	x	x	x	xx

beginloop — Begin loop — beginloop

Operation: **Uses:** –
Syntax: `beginloop` CIMM **Sets:** –

Description: (Calls internal kasm function.) A controller loop includes a BeginLoop with a controller immediate as argument, which pushes a new counter onto the counter stack. The matching EndLoop will decrement the counter and branch back if the counter is not -1.
Comments: The ASSEMBLER will automatically decriment your controller immediate value so that the number of iterations is the same as the value of the controller immediate. This decriment will be removed with the controller redesign, in which the counter comparison will be to 0 rather than -1.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	rrrrrrrrrrrrrrrr	x	x	x	x	x	x	x	xx	x	x	xx	0	0	0	1	x	x	xx

beginloopscr — Begin loop scratch — beginloopscr

Operation: **Uses:** –
Syntax: `beginloopscr` **Sets:** –

Description: (Calls internal kasm function.) An assembler pseudo-instruction (no machine code is generated) for beginning a loop with a preloaded loop counter. The assembler will match this with a nested EndLoop. To preload the counter from, for example, the input queue, perform: qintoscr, cntpush 0 ; nop ; scrtocntlo, qintoscr ; nop ; scrtocnthi, beginLoopScr ; ... ; endLoop. The matching EndLoop will decrement the counter and branch back if the counter is not -1.
Comments: WARNING: The controller redesign will include modifying the branch condition to the expected branch if counter is not 0. This will modifiy the semantics of this instruction because the decriment applied by the assembler with beginLoop is not applied to beginLoopScr. A beginLoopScr will iterate one more time than the value loaded into the loop counter. In the above example, the NOPs are required because the scratch register is always written at the start of the next instruction. This is because the scratch register is normally used for array outputs.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	x	x	x	x	x	xx	x	x	xx	x	x	x	x	x	x	xx

cbstoscr — Controller BS to scratch — cbstoscr

Operation: Scr ← CBS **Uses:** –
Syntax: `cbstoscr` **Sets:** –

Description: Copy the controller's bit shifter register to the scratch register.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	x	x	x	x	1	11	x	x	xx	x	x	x	x	x	x	xx

cntpush — Counter push — cntpush

Operation: Count ← ContImm **Uses:** –
Syntax: `cntpush` CImm **Sets:** –

Description: Push the 16-bit controller immediate onto the counter stack. This is identical to a BeginLoop except that the assembler does not generate a looping label

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	rrrrrrrrrrrrrrrr	x	x	x	x	x	x	x	xx	x	x	xx	0	0	0	1	x	x	xx

endloop — End loop — endloop

Operation: **Uses:** –
Syntax: `endloop` Label **Sets:** –

Description: (Calls internal kasm function.) A controller loop includes a BeginLoop with a controller immediate as argument, which pushes a new counter onto the counter stack. The matching EndLoop will decrement the counter and branch back if the counter is not -1.
Comments: WARNING: The semantics of this instruction will change to branch back if counter is not 0 after decrement in the controller redesign. This will not affect BeginLoop/EndLoop because of the automatic decrement, but will affect CNTPUSH and BEGINLOOPSCR.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	1111111111111111	x	x	x	x	x	x	x	xx	x	x	xx	0	1	1	0	x	x	xx

jdcntnz — Jump decrement counter not zero — jdcntnz

Operation: **Uses:** –
Syntax: `jdcntnz` LABEL **Sets:** –

Description: Jump if the controller's top of counter stack is not zero before the decrement.
Comments: WARNING: The semantics of this instruction will change to branch back if counter is not 0 after decrement in the controller redesign. Identical to endLoop, but without the assembler matching the label up with the appropriate beginLoop.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	1111111111111111	x	x	x	x	x	x	x	xx	x	x	xx	0	1	1	0	x	x	xx

jump — Jump — jump

Operation: **Uses:** –
Syntax: `jump` LABEL **Sets:** –

Description: Jump to the specified label.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	1111111111111111	x	x	x	x	x	x	x	xx	x	x	xx	0	x	x	x	0	0	01

jumpwor — Jump on wired or — jumpwor

Operation: **Uses:** –
Syntax: `jumpwor` LABEL **Sets:** –

Description: Jump to the specified label if the array's wired-or is 1.
Comments: There may be unresolved timing issues with this instruction.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	1111111111111111	x	x	x	x	x	x	x	xx	x	x	xx	1	x	x	x	0	0	00

qtoarr — Queue In to Array — qtoarr

Operation: ARRAY ← QIN **Uses:** –
Syntax: `qtoarr` DEST **Sets:** –

Description: Use the next input queue value as the array input. If the destination register is a left register, the input value is written to the rightmost register bank. If the destination is a right register, the input value is written to the left register bank.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	rxxxxx	xxxxxxxx

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	x	1	1	1	x	xx	x	x	xx	x	x	x	x	x	x	xx

qtoscr — Queue in to scratch — qtoscr

Operation: SCR ← QIN **Uses:** –
Syntax: `qtoscr` **Sets:** –

Description: Place a byte from the input queue in the controller's scratch register.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	x	x	x	x	1	10	x	x	xx	x	x	x	x	x	x	xx

scrtoarr — Scratch to Array — scrtoarr

Operation: Array ← Scratch **Uses:** –

Syntax: `scrtoarr` DEST **Sets:** –

Description: Use the controller's scratch register as the array input. If the destination register is a left register, the input value is written to the rightmost register bank. If the destination is a right register, the input value is written to the left register bank.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	rxxxxx	xxxxxxxx

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	x	1	0	0	x	xx	x	x	xx	x	x	x	x	x	x	xx

scrtoimm — Scratch to immediate — scrtoimm

Operation: **Uses:** –

Syntax: `scrtoimm` **Sets:** –

Description: Replace the array immediate field with the controller's scratch register. Can also be expressed as #scr wherever an immediate is used.

Comments: Do not change the scratch register on the previous instruction.

fr	alu	c	m	l	f	flag	bit	res	sram		opB	opA	opC	dest	array
ce	func	i	p	c	i	bus	shift	mx	r	w	sel	register	register	register	immediate
x	xxxxx	x	x	x	x	xxxxx	xxxx	xx	x	x	xxx	xxxxxx	xxxxxx	xxxxxx	00000000

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt		pc		
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	0	xxx	x	xxxxxxxxxxxxxxxx	x	x	x	x	x	x	x	xx	x	x	11	x	x	x	x	x	x	xx

15 Controller Modifiers

breakpoint — **Breakpoint** — breakpoint

Operation: **Uses:** –
Syntax: `breakpoint` **Sets:** –

Description: Set a program breakpoint.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt			pc	
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	1	x	x	x	x	x	x	xx	x	x	xx	x	x	x	x	x	x	xx

cbsload — **Controller BS load** — cbsload

Operation: **Uses:** –
Syntax: `cbsload` **Sets:** –

Description: Load the controller's Bitshifter with the 8 wired-OR signals.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt			pc	
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	x	x	x	x	x	xx	x	1	xx	x	x	x	x	x	x	xx

cbssleft — **Controller BS shift** — cbssleft

Operation: **Uses:** –
Syntax: `cbssleft` **Sets:** –

Description: Shift a wired-or bit onto the low bit of the controller's bitshifter.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt			pc	
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	x	x	x	x	x	xx	1	x	xx	x	x	x	x	x	x	xx

forceqout — **Force Qout** — forceqout

Operation: **Uses:** –
Syntax: `forceqout` **Sets:** –

Description: Force write to output queue regardless of array mask output.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt			pc	
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	1	x	x	x	x	x	xx	x	x	xx	x	x	x	x	x	x	xx

scrtocnthi — **Scratch to Counter High** — scrtocnthi

Operation: COUNTER[8:15] ← SCRATCH **Uses:** –
Syntax: `scrtocnthi` **Sets:** –

Description: Copy the controller's scratch register to the high byte of the controller's top of counter stack register.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt			pc	
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	x	x	x	x	x	xx	x	x	10	x	x	x	x	x	x	xx

scrtocntlo — Scratch to Counter Low — scrtocntlo

Operation: COUNTER[0:7] ← SCRATCH **Uses:** –

Syntax: `scrtocntlo` **Sets:** –

Description: Copy the controller's scratch register to the low byte of the controller's top of counter stack register.

spa	im	diag	D	controller	br	f	i/o		dsel		scratch		cbs		imm	br		cnt			pc	
re	mx	out	pop	immediate	k	ot	r	w	r	l	w	mx	sh	ld	sel	w	c	d	ps	po	pu	sel
xx	x	xxx	x	xxxxxxxxxxxxxxxx	x	x	x	x	x	x	x	xx	x	x	01	x	x	x	x	x	x	xx

16 Assembler Directives

BEGINCOND — Begin conditional — BEGINCOND

Operation: **Uses:** –
Syntax: `BEGINCOND` **Sets:** –

Description: (Calls internal kasm function.) BeginCond/EndCond pairs indicate nesting of conditionals. While inside a conditional section, the Kestrel force bit is by default turned off. Code can be forced within a BeginCond/EndCond with the FORCE modifier.
Comments: BeginCond and EndCond do not generate any machine code.

DEFINE — Define symbol — DEFINE

Operation: **Uses:** –
Syntax: `DEFINE` **Sets:** –

Description: (Calls internal kasm function.) Define a KASM symbol, as in 'define NumPE 512'. The text is added to the symbol table and can be referred to by $NumPE or $NumPE$. No white space is introduced in the expansion. Symbols are case-insensitive and may only include letters and underscores.
Comments: Define must occur on a line by itself. KASM does not support static expression evaluation but resolves nested definitions as in 'define NumData $NumPE'. There is a compiled limit of 5000 symbols including active macro parameters and definitions. Definitions and labels have dynamic scope: if X is defined in macro M1, and M1 calls macro M2, X is defined in M2.

ENDCOND — End conditional — ENDCOND

Operation: **Uses:** –
Syntax: `ENDCOND` **Sets:** –

Description: (Calls internal kasm function.) BeginCond/EndCond pairs indicate nesting of conditionals. While inside a conditional section, the Kestrel force bit is by default turned off. Code can be forced within a BeginCond/EndCond with the FORCE modifier.
Comments: BeginCond and EndCond do not generate any machine code.

INCLUDE — Include KASM file — INCLUDE

Operation: **Uses:** –
Syntax: `INCLUDE` **Sets:** –

Description: (Calls internal kasm function.) Read a KASM file into the assembler.

MACRODEF — Macro definition — MACRODEF

Operation: **Uses:** –
Syntax: `MACRODEF` **Sets:** –

Description: (Calls internal kasm function.) MacroDef/MacroEnd pairs indicate the definition of a macro. A statement of the form 'MacroDef MacName (p1,p2,p3)' must be on a line by itself. The macro MacName is entered into the symbol table along with its parameters (if there are no parameters, both left and right parenthesis are still required). The lines of code between the MacroDef and MacroEnd (on its own line) are stored for future use, including any present including 'include' or 'define' statements. A macro cannot be defined within another macro.
To use a macro, put, for example, 'MacName (L1, L2, #37)' on a line by itself or with a comment. At the start of the macro, an implicit define for each parameter will be performed, and a substitution will be made for the formal parameters when required. As with other define statements, formal parameter names must be preceded with a dollar sign.
For various reasons, KASM uses dynamic scoping. That is, if a symbol is not defined in the current context (ie, the current macro being parsed), it is searched for in the macro calling stack back up to the global context.
It is highly recommended that this feature be used with care; passing parameters (e.g., labels) down the macro call stack is greatly preferred over relying on labels defined based on dynamic scope.
Arguments to macros that are themselves defined (including labels) are evaluated within the context of the calling code.
A macro can call another macro. The KASM assembler will keep track of a sequence of nested macro calls in its error messages and debugging line numbers so that macro use can be traced. An error line number of the form 'x.kasm:5:x.kasm:20' indicates that an error occured in the macro's line 5, where the macro was called from line 20.
Comments: Limits of 50 parameters and 1000 macros are compiled into the assembler.

MACROEND — End macro definition — MACROEND

Operation: **Uses:** –
Syntax: `MACROEND` **Sets:** –

Description: (Calls internal kasm function.) End a macro definition. See MacroDef.

Index