

## Arithmetic Operations

	Short Description:	Definition:	Full Description:
<b>ADD</b>	Unsigned integer add	ADD REG1, REG2, OP(255)	Performs 32-bit add on two 32-bit zero extended source values
<b>ADC</b>	Unsigned integer add (carry)	ADC REG1, REG2, OP(255)	Performs 32-bit add on two 32-bit zero extended source values, plus a stored carry bit
<b>SUB</b>	Unsigned integer subtract	SUB REG1, REG2, OP(255)	Performs 32-bit subtract on two 32-bit zero extended source values
<b>SUC</b>	Unsigned integer subtract (carry)	SUC REG1, REG2, OP(255)	Performs 32-bit subtract on two 32-bit zero extended source values with carry (borrow)
<b>RSB</b>	Reverse unsigned int subtract	RSB REG1, REG2, OP(255)	Performs 32-bit subtract on two 32-bit zero extended source values. Source values reversed
<b>RSC</b>	Reverse unsigned integer subtract (carry)	RSC REG1, REG2, OP(255)	Performs 32-bit subtract on two 32-bit zero extended source values with carry (borrow). Source values reversed

## Logical Operations

<b>LSL</b>	Logical shift left	LSL REG1, REG2, OP(31)	Performs 32-bit shift left of the zero extended source value
<b>LSR</b>	Logical shift right	LSR REG1, REG2, OP(31)	Performs 32-bit shift right of the zero extended source value
<b>AND</b>	Bitwise AND	AND REG1, REG2, OP(255)	Performs 32-bit logical AND on two 32-bit zero extended source values
<b>OR</b>	Bitwise OR	OR REG1, REG2, OP(255)	Performs 32-bit logical OR on two 32-bit zero extended source values
<b>XOR</b>	Bitwise XOR	XOR REG1, REG2, OP(255)	Performs 32-bit logical XOR on two 32-bit zero extended source values
<b>NOT</b>	Bitwise NOT	NOT REG1, REG2	Performs 32-bit logical NOT on the 32-bit zero extended source value
<b>MIN</b>	Copy minimum	MIN REG1, REG2, OP(255)	Compares two 32-bit zero extended source values and copies the smaller to REG1
<b>MAX</b>	Copy maximum	MAX REG1, REG2, OP(255)	Compares two 32-bit zero extended source values and copies the larger to REG1
<b>CLR</b>	Clear bit	CLR REG1, REG2, OP(31)	Clears the specified bit in the source and copies the result to the destination Also: <b>CLR REG1, OP(31)</b> <b>CLR REG1, Rn.tx</b> <b>CLR Rn.tx</b>
<b>SET</b>	Set bit	SET REG1, REG2, OP(31)	Sets the specified bit in the source and copies the result to the destination Also: <b>SET REG1, OP(31)</b> <b>SET REG1, Rn.tx</b> <b>SET Rn.tx</b>
<b>SCAN</b>	Register field scan	SCAN Rn, OP(255)	The SCAN instruction scans the register file for a particular value. It includes a configurable field width and stride. The width of the field to match can be set to 1, 2, or 4 bytes.
<b>LMBD</b>	Left-most bit detect	LMBD REG1, REG2, OP(255)	Scans REG2 from its left-most bit for a bit value matching bit 0 of OP(255), and writes the bit number in REG1 (writes 32 to REG1 if the bit is not found)

## Register Load and Store

<b>MOV</b>	Copy value	MOV REG1, OP(65535)	Moves the value from OP(65535), zero extends it, and stores it into REG1
<b>LDI</b>	Load immediate	LDI REG1, IM(65535)	The LDI instruction moves value from IM(65535), zero extends it, and stores it into REG1
<b>MVIB</b>	Move register file indirect (8)	MVIB [*&]REG1, [*&]REG2	The MVx instruction family moves a value from the source to the destination. The source, destination, or both can be register pointers.
<b>MVIW</b>	Move register file indirect (16)	MVIW [*&]REG1, [*&]REG2	
<b>MVID</b>	Move register file indirect (32)	MVID [*&]REG1, [*&]REG2	
<b>LBBO</b>	Load byte burst	LBBO REG1, Rn2, OP(255), IM(124) LBBO REG1, Rn2, OP(255), bn	The LBBO instruction is used to read a block of data from memory into the register file. The memory address to read from is specified by a 32-bit register, using an optional offset
<b>SBBO</b>	Store byte burst	SBBO REG1, Rn2, OP(255), IM(124) SBBO REG1, Rn2, OP(255), bn	The SBBO instruction is used to write a block of data from the register file into memory. The memory address to which to write is specified by a 32-bit register, using an optional offset
<b>LBCO</b>	Load byte burst with constant table offset	LBCO REG1, Cn2, OP(255), IM(124) LBCO REG1, Cn2, OP(255), bn	The LBCO instruction is used to read a block of data from memory into the register file. The memory address from which to read is specified by a 32-bit constant register (Cn2), using an optional offset from an immediate or register value
<b>SBCO</b>	Store byte burst with constant table offset	SBCO REG1, Cn2, OP(255), IM(124) SBCO REG1, Cn2, OP(255), bn	The SBCO instruction is used to write a block of data from the register file into memory. The memory address to write to is specified by a 32-bit constant register (Cn2), using an optional offset from an immediate or register value
<b>ZERO</b>	Clear register space	ZERO IM(123), IM(124)	This pseudo-op is used to clear space in the register file. Also: <b>ZERO &amp;REG1, IM(124)</b>

## Program Flow Control

<b>JMP</b>	Unconditional jump	JMP OP(65535)	Unconditional jump to a 16-bit instruction address, specified by register or immediate value
<b>JAL</b>	Unconditional jump and link	JAL REG1, OP(65535)	Unconditional jump to a 16-bit instruction address, specified by register or immediate value. Address following the JAL instruction is stored into REG1, so that REG1 can later be used as a "return" address
<b>CALL</b>	Call procedure	CALL OP(65535)	The CALL instruction is designed to emulate a subroutine call on a stack-based processor
<b>RET</b>	Return from procedure	RET	The RET instruction is designed to emulate a subroutine return on a stack-based processor
<b>QBGT</b>	Quick branch if >	QBGT LABEL, REG1, OP(255)	Jumps if the value of OP(255) is greater than REG1
<b>QBGE</b>	Quick branch if ≥	QBGE LABEL, REG1, OP(255)	Jumps if the value of OP(255) is greater than or equal to REG1
<b>QBLT</b>	Quick branch if <	QBLT LABEL, REG1, OP(255)	Jumps if the value of OP(255) is less than REG1
<b>QBLE</b>	Quick branch if ≤	QBLE LABEL, REG1, OP(255)	Jumps if the value of OP(255) is less than or equal to REG1
<b>QBEQ</b>	Quick branch if =	QBEQ LABEL, REG1, OP(255)	Jumps if the value of OP(255) is equal to REG1
<b>QBNE</b>	Quick branch if ≠	QBNE LABEL, REG1, OP(255)	Jumps if the value of OP(255) is NOT equal to REG1
<b>QBA</b>	Quick branch always	QBA LABEL	Jump always. This is similar to the JMP instruction, only QBA uses an address offset and thus can be relocated in memory
<b>QBBS</b>	Quick branch if bit is set	QBBS LABEL, REG1, OP(31)	Jumps if the bit OP(31) is set in REG1. Also: <b>QBBS LABEL, Rn.tx</b>
<b>QBBC</b>	Quick branch if bit is clear	QBBC LABEL, REG1, OP(31)	Jumps if the bit OP(31) is clear in REG1. Also: <b>QBBC LABEL, Rn.tx</b>
<b>WBS</b>	Wait until bit set	WBS REG1, OP(31) WBS Rn.tx	The WBS instruction is a pseudo op that uses the QBBC instruction. It is used to poll on a status bit, spinning until the bit is set
<b>WBC</b>	Wait until bit clear	WBC REG1, OP(31) WBC Rn.tx	The WBC instruction is a pseudo op that uses the QBBS instruction. It is used to poll on a status bit, spinning until the bit is clear
<b>HALT</b>	Halt operation	HALT	The HALT instruction disables the PRU. This instruction is used to implement software breakpoints in a debugger
<b>SLP</b>	Sleep operation	SLP IM(1)	The SLP instruction will sleep the PRU, causing it to disable its clock. This instruction can specify either a permanent sleep or a "wake on event"

See: [http://processors.wiki.ti.com/index.php/PRU\\_Assembly\\_Instructions](http://processors.wiki.ti.com/index.php/PRU_Assembly_Instructions) for further information.

REG, REG1, REG2, ...	A register field from 8 to 32 bits	bn	A field that must be b0 to b3
Rn, Rn1, Rn2, ...	A 32-bit register field (r0 to r31)	LABEL	A valid label
Rn.tx	A 1-bit register field	IM(n)	An immediate value from 0 to n
Cn, Cn1, Cn2, ...	A 32-bit constant constant register (c0 to c31)	OP(n)	Operand - either a REG or IM(n)