Registers

Register	Description
R0	16 bit, General Purpose
R1	16 bit, General Purpose
R2	16 bit, General Purpose
R3	16 bit, General Purpose
R4	16 bit, General Purpose
R5	16 bit, General Purpose
R6	16 bit, General Purpose
R7	16 bit, General Purpose
SP	16 bit, Stack Pointer
PC	16 bit, Program Counter

All registers are 16 bit. ALU operations are 16 bit. 8 bit operations are not natively supported except for extended loads and truncated stores. Registers R0 through R7 are general purpose. Registers SP and PC do not belong to the general set, so they use special purpose instructions.

Status Register

Register	Description
Status I T C Z	Status Register I: Interrupt flag T: Condition flag, result of a compare instruction C, Z: Carry, Zero flags, result of

- Compare instructions compare two operands for a specified condition code and set T to 1 if the condition was met or 0 otherwise, the C and Z flags are set according to a normal subtraction.
- ALU arithmetic and logical instructions set C and Z according to the result. Additionally, the Z flag is copied to T. For example, the add instruction will set C to 1 if there was a carry, and both Z and T to 1 if the result was zero.
- Conditional instructions such as setcc, selcc and brcc take the T flag as the condition to watch

Condition Codes Summary

Encoding	Machine Name	Alt Names	SR Flags	Description
000	eq	z	Z	Equal than. Zero
001	ne	nz	!Z	Not equal. Not zero
010	uge	hs, c	C	Unsigned greater than or equal. Carry
011	ult	lo, nc	!C	Unsigned less than. Not carry
100	ge	-	S == V	Signed greater than or equal
101	lt	-	S != V	Signed less than
110	ugt	hi	C && !Z	Unsigned greater than
111	gt	-	(S == V) && !Z	Signed greater than
-	ule	ls	!C || Z	Unsigned less than or equal Implemented as the opposite of ugt
-	le	-	(S != V) || Z	Signed less than or equal Implemented as the opposite of gt

The S and V flags are computed internally to match condition codes, but they are not stored in the status register, or are available to the user.

Instruction Formats

Type	Encoding Fields					Description
P	opcode (5)	immediate (11)				Long immediate
I2	opcode (5)	Rd (3)	Rs (3)	immediate (5)		Two registers with immediate
I1	opcode (5)	Rd (3)	immediate (8)			One register with immediate
J	opcode (5)	(2)	immediate (9)			No registers with immediate
R3	opcode (5)	Rd (3)	Rs (3)	(2)	Rn (3)	Three registers
R2	opcode (5)	Rd (3)	Rs (3)	opcode (5)		Zero, One or Two registers

Opcodes Summary

Type	Encoding Bits																Description
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
P	op (11100..11111)					aaa aaaa aaaa											Prefix, call immediate
I2	op (10100..11011)					Rd/cc			Rs			k kkkk					Load/store/lea with register+immediate, And/compare with immediate
I1	op (01000..10011)					Rd			kkkk kkkk								Move/add/sub immediate, Load/store indirect, Load/store from stack
J	0	0	op (11000..11111)					a aaaa aaaa									Conditional/unconditional branch, Add SP
R3	0	0	op (010..101)			Rd/cc			Rs			Rn			oo		Three register ALU operation, Select Load/store with register offset
R2	0	0	0	0	x	Rd/xxx			Rs/xxx			op (00000..11111)					Two register ALU operation, Move, Setcc, Jump/Call Indirect, Zero Operand Instructions
	0	0	0	0	0	000			000			0	0	0	0	0	NOP instruction, emulated through 'mov r0, r0'

Instruction Decoding

Type	Encoding Bits																Description
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
P,I2,I1	en2 01..11		x														Primary opcode is 10 (*) Secondary opcode is taken from bits 11 to 15
J	en1 0011				x												Primary opcode is 01 (*) Secondary opcode is from bits 0 to 1, 11 to 13
R3	en1 0001..0010				x												Primary opcode is 01 (*) Secondary opcode is taken from bits 9 to 13
R2	en0 0000				x												Primary opcode is 00 (*) Secondary opcode is taken from bits 7 to 11

(*) 7 bit condensed instruction encodings are made by combining 2 bit primary opcodes with 5 bit secondary opcodes

Immediate Fields Decoding

Type	Encoding Bits																Description
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
P	imm3 111			x		aaa aaaa aaaa											zero extend 11 bit immediate
I2	imm2 101...110			x								k kkkk					zero extended 5 bit immediate sign extend if instruction opcode is 1010x
I1	imm1 010...100			x					kkkk kkkk								zero extended 8 bit immediate sign extend if instruction opcode is 01100
J	imm0 00		11		x			a aaaa aaaa									sign extend 9 bit immediate (1)
			x					-									

(*) The lower 5 bits of immediate constants are always encoded in bits 6 to 10. The most significant bits appear in the remaining encoding bits.
(1) J Type immediates are lazily decoded based only on the first 2 encoding bits.The decoder will provide unused bogus immediates for instructions of type R2, R3.

Prefixed instructions

The prefixed instructions are assembler emulated instructions that are made of core instructions preceded by a prefix instruction. The prefix instruction contains a 'p_imm' 11 bit immediate field that expands the functionality of core instructions. The prefix instruction extends the immediate field 'imm' of the next instruction by replacing it with the result of the logical expression: (p_imm << 5) | (imm & 0b11111), thus providing a full 16 bit immediate range to the prefixed instruction.

The following *non exhaustive list* shows several examples of prefix instruction transformations:

Core Instruction	Prefix	Prefixed Instruction	Description
Arithmetic, Logic			
add Rd, K, Rd	pfix_k	add Rd, #K, Rd	Add with long immediate. The 8 bit embedded immediate is replaced by a 16 bit one
and Rd, K, Rd	pfix_k	and Rd, #K, Rd	And with long immediate. The 8 bit embedded immediate is replaced by a 16 bit one
lea Rs, K, Rd	pfix_k	addx Rs, #K, Rd	Lea with long immediate. The 5 bit embedded immediate is replaced by a 16 bit one
Moves			
mov K, Rd	pfix_k	mov #K, Rd	Copy K into Rd. The 8 bit embedded immediate is replaced by a 16 bit one
Branching and subroutines			
br%cc Label	pfix_k	br%cc Label	Conditional branch. Branch instruction reach is extended from 9 to 16 bit long offsets
call Label	pfix_k	call &Label	Subroutine call. Call instruction reach is extended from 11 to 16 bit addresses
Memory			
ld.w [Rs, K], Rd	pfix_k	ld.w [Rs, #K], Rd	Load word with immediate offset. The 5 embedded immediate is replaced by a 16 bit one
ld.w [A], Rd	pfix_k	ld.w [&A], Rd	Load word with immediate absolute address. The 8 bit embedded immediate field is replaced by a 16 bit address

Carry-in instructions

A number of instructions take the carry flag to enable wider than native operations. For example, a 32 bit addition can be performed on two pairs of registers representing 32 bit values, by sequentially executing 'add' on the lower register operands, followed by an 'addc' on the upper register operands.

The following carry-in instructions are available:

addc Rs, Rn, Rd	Add with carry
subc Rs, Rn, Rd	Subtract with carry
cmpc Rs, Rn	Compare with carry
lsrc Rs, Rd	Shift right through carry

Carry-in instructions are designed to be executed in combination with carry setting instructions of the same family. The Status Register flags after carry-in instructions will correctly reflect the result of the combined operation. Therefore it is safe to use conditional branch or move instructions after them.

Addressing modes

Name	Instr Types	Example Instructions	Description
Implicit	R2	ret	The instruction has no explicit operands
Register with Immediate	I1, I2, J	add Rd, K, Rd addx Rs, K, Rd addx,K SP	The immediate constant value is presented as a bit field in the instruction encoding
Register	R2, R3	mov Rs, Rd add Rs, Rn, Rd	The instruction contains register operands
Absolute Address	P	call &Label	The instruction contains an absolute memory address
PC Relative Address	J	brcc Label	The instruction contains a PC relative memory address
Indirect Absolute	I1	ld.sb [&A], Rd	The instruction contains the memory address of the referred value
Indirect Base Register plus Offset	I1, I2	ld.sb [SP, K], Rd ld.sb [Rs, K], Rd	The instruction specifies a base register and an offset which is added to the base to form the memory address of the referred value
Indirect Base Register plus Index	R3	ld.sb [Rs, Rn], Rd	The instruction specifies a base register and an index register which are added to form the memory address of the referred value
Indirect Base Register plus scaled Index	R3	ld.w [Rs, Rn], Rd	The instruction specifies a base register and an index register. The index register value is scaled by 2 and added to the base to form the memory address of the referred value

Scaled memory accesses

All word memory accesses are two bytes aligned. This property is implicitly used on all word memory access instructions by scaling immediates and register indices by 2. Therefore:

- The least significative bit of constant immediates is not encoded, and it is interpreted to be always zero.
- Register index values are scaled by two (shifted one bit left) before using.

Instructions Summary

The following table shows the complete instruction list, separated by functional categories:

Category	Assembly Mnemonic	Description
Arithmetic, Logic		
Arithmetic	add Rd, K, Rd add Rs, Rn, Rd	Add
	addx Rs, K, Rd addx SP, K, Rd addx SP, K, SP	Non flag altering Add
	addc Rs, Rn, Rd	Add with carry
	sub Rd, K, Rd sub Rs, Rn, Rd	Subtract
	subc Rs, Rn, Rd	Subtract with carry
	neg Rs, Rd	Negate
Logic	and Rs, K, Rd and Rs, Rn, Rd	Logical And
	or Rs, Rn, Rd	Logical Or
	xor Rs, Rn, Rd	Exclusive logical Or
	not Rs, Rd	Logical Not
Shifts	asr Rs, Rd	Arithmetic shift right
	lsr Rs, Rd	Logical shift right
	lsrc Rs, Rd	Logical shift right through carry
Byte shifts and extensions	asrb Rs, Rd	8 bit arithmetic shift right
	lsrb Rs, Rd	8 bit logical shift right
	lslb Rs, Rd	8 bit logical shift left
	zext Rs, Rd	Zero extend byte
	sext Rs, Rd	Sign extend byte
	sextw Rs, Rd	Sign extend word
Comparison	cmp.%cc Rd, K cmp.%cc Rs, Rn	Compare
	cmpc.%cc Rd, K cmpc.%cc Rs, Rn	Compare with carry
Data moves		
Moves	mov K, Rd mov Rs, Rd mov Rs, SP	Move
Conditional moves	selcc Rs, Rn, Rd selcc 0, Rs, Rd selcc Rs, 0, Rd	Select
	setcc Rd setncc Rd	Set
Branching and subroutines		
Branch instructions	jmp Label jmp Rs	Unconditional branch
	brcc Label brncc Label	Conditional branch
Subroutine instructions	call Label call Rd	Call to subroutine

Category	Assembly Mnemonic	Description
	ret	Return from subroutine
Memory access		
Memory load	ld.w [Rs, K], Rd ld.w [Rs, Rn], Rd ld.w [&A], Rd ld.w [SP, K], Rd	Load word from memory
	ld.w {Rs}, Rd	Load word from program memory
	ld.sb [Rs, K], Rd ld.sb [Rs, Rn], Rd ld.sb [&A], Rd ld.sb [SP, K], Rd	Load sign extended byte from memory
	ld.zb [Rs, Rn], Rd	Load zero extended byte from memory
Memory store	st.w Rd, [Rs, K] st.w Rd, [Rs, Rn] st.w Rd, [&A] st.w Rd, [SP, K]	Store word to memory
	st.b Rd, [Rs, K] st.b Rd, [Rs, Rn] st.b Rd, [&A] st.b Rd, [SP, K]	Store byte to memory
Interrupts		
Interrupt instructions	dint	Disable interrupt
	eint	Enable interrupt
	reti	Return from interrupt
	halt	Halts processor

Instructions Summary, by opcode

Type	Opcode	Machine Name	Assembly Mnemonic	Description
Two register Move, ALU operation				
R2	00000	mov_rr	mov Rs, Rd	Copy Rs to Rd
	00001	mov_rq	mov Rs, SP	Copy Rs to SP
	00010	zext_rr	zext Rs, Rd	Move zero-extended Rs low byte to Rd
	00011	sext_rr	sext Rs, Rd	Move sign-extended Rs low byte to Rd
	00100	lsrb_rr	lsrb Rs, Rd	8 bit logical shift right
	00101	asrb_rr	asrb Rs, Rd	8 bit arithmetic shift right
	00110	lslb_rr	lslb Rs, Rd	8 bit logical shift left
	00111	sextw_rr	sextw Rs, Rd	Sets Rd to all ones if Rs is negative, or zero otherwise
Two Register ALU Operation				
R2	01000	lsr_rr	lsr Rs, Rd	Logical shift right. Bit 0 is shifted to the C Flag. Bit 15 is set to zero.
	01001	lsrc_rr	lsrc Rs, Rd	Shift Right through carry. Bit 0 is shifted to the C Flag. The old C flag is shifted to bit 15
	01010	asr_rr	asr Rs, Rd	Arithmetic shift right. Bit 0 is shifted to the C Flag. bit 15 is preserved
	01011	movw_pr	ld.w {Rs}, Rd	Load Program Memory
	01100	sel_0rr	selcc 0, Rs, Rd	Conditional set. Move Rs to Rd if T flag is not set, otherwise move 0 to Rd
	01101	sel_r0r	selcc Rs, 0, Rd	Conditional set. Move Rs to Rd if T flag is set, otherwise move 0 to Rd
	01110	neg_rr	neg Rs, Rd	Rd = 0 - Rs, update SR
	01111	not_rr	not Rs, Rd	Rd = ~Rs, update SR
				(*) Left shifts are implemented with the add and addc instructions
Branch/Call indirect, Setcc				
R2	10000	jmp_r	jmp Rs	Jump to Rs
	10001	call_r	call Rd	Subroutine call to Rd
	10010	-	-	Reserved
	10011	-	-	Reserved
	10100	mov_sr	mov SR, Rd	Copy Status Register to Rd (Not implemented)
	10101	mov_rs	mov Rs, SR	Restore Status Register from Rs (Not implemented)
	10110	set_nt	setncc Rd	Conditional set. Move 1 to Rd if T flag is not set, otherwise move 0 to Rd
	10111	set_t	setcc Rd	Conditional set. Move 1 to Rd if T flag is set, otherwise move 0 to Rd
Zero Operand Instructions				
R2	11000	ret	ret	Return from subroutine
	11001	reti	reti	Return from interrupt
	11010	dint	dint	Disable interrups
	11011	eint	eint	Enable interrupts
	11100	halt	halt	Halts processor and sets it into program mode
	11101	-	-	Reserved
	11110	-	-	Reserved
	11111	-	-	Reserved

Type	Opcode	Machine Name	Assembly Mnemonic	Description
Three register ALU operation				
R3	01000	cmp_crr	cmp.%cc Rs, Rn	Compare Rs with Rn and update SR flags
	01001	cmpc_crr	cmpc.%cc Rs, Rn	Compare Rs with Rn and update SR flags
	01010	subc_rrr	subc Rs, Rn, Rd	Rd = Rs - (Rn+C), update SR
	01011	sub_rrr	sub Rs, Rn, Rd	Rd = Rs - Rn, update SR
	01100	and_rrr	and Rs, Rn, Rd	Rd = Rs & Rn, update SR
	01101	or_rrr	or Rs, Rn, Rd	Rd = Rs | Rn, update SR
	01110	sel_rrr	selcc Rs, Rn, Rd	Conditional select. Copy Rs to Rd if T flag is set otherwise copy Rn to Rd
	01111	-	-	Reserved
Load/store with register offset				
R3	10000	xor_rrr	xor Rs, Rn, Rd	Rd = Rs ^ Rn, update SR
	10001	adc_rrr	addc Rs, Rn, Rd	Rd = Rs + (Rn+C), update SR
	10010	add_rrr	add Rs, Rn, Rd	Rd = Rs + Rn, update SR
	10011	movw_nr	ld.w [Rs, Rn], Rd	Load word at aligned memory address Rs+Rn, store in Rd
	10100	movzb_nr	ld.zb [Rs, Rn], Rd	Load byte at memory address Rs+Rn, store in Rd
	10101	movsb_nr	ld.sb [Rs, Rn], Rd	Load byte at memory address Rs+Rn, store in Rd
	10110	movw_rn	st.w Rd, [Rs, Rn]	Store Rd in word aligned memory address Rs+Rn
	10111	movb_rn	st.b Rd, [Rs, Rn]	Store lower byte of Rd in memory address Rn+Rs
Conditional/unconditional branch, Add SP				
J	11000	-	-	Reserved
	11001	-	-	Reserved
	11010	-	-	Reserved
	11011	-	-	Reserved
	11100	br_nt	brncc Label	Conditional PC relative branch if T flag is not set, otherwise proceed with the next instruction
	11101	br_t	brcc Label	Conditional PC relative branch if T flag is set, otherwise proceed with the next instruction
	11110	add_kq	addx SP, K, SP	Add signed immediate to SP
	11111	jmp_k	jmp Label	Unconditional PC relative branch to Label
	(*) 'Label' and 'K' are 9 bit signed immediates extensible to a 16 bit word with the prefix instruction			
Move/add/sub immediate, Load/store indirect, Load/store from stack				
I1	01000	movw_ar	ld.w [&A], Rd	Load word at aligned memory address A, store in Rd
	01001	movsb_ar	ld.sb [&A], Rd	Load byte at memory address A, sign-extend into Rd
	01010	movw_ra	st.w Rd, [&A]	Store Rd in word aligned memory address A
	01011	movb_ra	st.b Rd, [&A]	Store lower byte of Rd in memory address A
	01100	mov_kr	mov K, Rd	Copy sign-extended K into Rd
	01101	sub_kr	sub Rd, K, Rd	Subtract zero-extended K from Rd, store in Rd, update SR
	01110	add_kr	add Rd, K, Rd	Add zero-extended K to Rd, store result in Rd, update SR
	01111	lea_qr	addx SP, K, Rd	Add zero-extended K to SP, store result in Rd
	10000	movw_qr	ld.w [SP, K], Rd	Load word at aligned memory address SP+K, store in Rd.
	10001	movsb_qr	ld.sb [SP, K], Rd	Load byte at memory address SP+K, sign-extend into Rd
	10010	movw_rq	st.w Rd, [SP, K]	Store Rd in word aligned memory address SP+K

Type	Opcode	Machine Name	Assembly Mnemonic	Description
	10011	movb_rq	st.b Rd, [SP, K]	Store lower byte of Rd in memory address SP+K
				(*) 'K' is a 8 bit immediate in the range 0..255, or -128..+127 extensible to a 16 bit word with the prefix instruction
Load/store with immediate offset				
I2	10100	cmp_crk	cmp.%cc Rs, K	Compare Rd with sign-extended K and update SR flag
I2	10101	cmpc_crk	cmpc.%cc Rs, K	Compare Rd with sign-extended K and update SR flag
I2	10110	and_kr	and Rs, K, Rd	AND zero-extended K with Rd, store in Rd, update SR
I2	10111	lea_mr	addx Rs, K, Rd	Add zero-extended K to Rs, store result in Rd
I2	11000	movw_mr	ld.w [Rs, K], Rd	Load word at aligned memory address Rn+K, store in Rd
I2	11001	movsb_mr	ld.sb [Rs, K], Rd	Load byte at memory address Rn+K, sign-extend into Rd
I2	11010	movw_rm	st.w Rd, [Rs, K]	Store Rd in word aligned memory address Rn+K
I2	11011	movb_rm	st.b Rd, [Rs, K]	Store lower byte of Rd in memory address Rn+K
				(*) 'K' is a 5 bit immediate in the range 0..31 extensible to a 16 bit word with the prefix instruction
Prefix, call immediate				
P	11100	-	-	Reserved
P	11101	-	-	Reserved
P	11110	call	call &Label	Call immediate
P	11111	pfix_k	pfix K	Prefix immediate
				(*) Label is a 11 bit immediate extensible to a 16 bit word with the prefix instruction